Evaluating the use of ICN for Internet of things

Johan Carlquist

January 21, 2018

Abstract

The market of IoT devices continues to grow at a rapid speed as well as constrained wireless sensor networks. Today, the main network paradigm is *host centric* where a users have to specify which host they want to receive their data from. Information-centric networking is a new paradigm for the future internet, which is based on *named data* instead of *named hosts*. With ICN, a user sends a request for a perticular data on the network, any router or server containing the data will respond to the request.

In order to achieve low latency between data creation and its consumption, as well as being able to follow the sequential production of data, an algorithm was created for these purposes during this thesis. This algorithm uses a 'one-time subscription' approach to initiate a *interest* message before the data has been created at the sensor.

The result of this algorithm shows that a consumer can retrieve the data with minimum latency from its creation by the sensor, without using a publish/subscribe system such as MQTT or similar. The study shows that it can handle larger variations of clock drift from the sensor. The performance evaluation carried out which analysed the Content Centric Network application on the sensor shows that the application has little impact on the overall round trip time in the network.

Based on the results, this thesis concluded that the ICN paradigm, together with a 'one-time subscription' model, can be a suitable option for communication within the IoT domain where consumers ask for sequentially produced data.

Acknowledgement

I would like to take this opportunity to sincerely thank my two supervisors, Bengt Ahlgren and Anders Lindgren from Research Institute of Sweden for their valuable support and input throughout the entire thesis project. Without their help and great knowledge, this project would not have succeeded.

I would also like to thank my mentor Edith Ngai for the feedback, discussions and relevent input of which made it possible to complete this thesis.

Furthermore, I would like to take the opportunity to thank Valtech Sweden for their support and help with computer equiment and more.

Last but not the least, I would like to thank Anna Rumetshofer who has helped with encouraging, read through and general advices regarding the writing part of the thesis.

Contents

1 Introduction **5**
- 1.1 Problem statement . . . 5
- 1.2 Delimitations . . . 6
- 1.3 Research methodology . . . 6
- 1.4 Contributions . . . 6
- 1.5 Related work . . . 7
- 1.6 Structure of the Report . . . 7

2 Background **9**
- 2.1 Internet of Things - network stack . . . 9
  - 2.1.1 IEEE 802.15.4 . . . 9
  - 2.1.2 IPv4, IPv6 . . . 10
  - 2.1.3 6LowPAN . . . 11
  - 2.1.4 MQTT, MQTT-SN . . . 11
  - 2.1.5 CoAP . . . 12
- 2.2 Information Centric Networking . . . 13
  - 2.2.1 Content-Centric Networking . . . 13
  - 2.2.2 Using ICN for IoT . . . 14
  - 2.2.3 CCN-lite . . . 15
- 2.3 Contiki-OS . . . 15
  - 2.3.1 Sparrow . . . 15
  - 2.3.2 CCN-lite portation . . . 16

3 Implementation **17**
- 3.1 Experimental Setup . . . 17
  - 3.1.1 Hardware . . . 17
  - 3.1.2 Software . . . 19
  - 3.1.3 Communication between gateway and sensor . . . 20
  - 3.1.4 Retrieve data . . . 20

4 Latency performance evaluation **21**
- 4.1 Method . . . 21
- 4.2 Results . . . 24

5 Suitability evaluation **28**
- 5.1 Method . . . 29
- 5.2 Result . . . 34
  - 5.2.1 Different intervals at the sensor . . . 38

6 Discussion **41**
- 6.1 Performance evaluation . . . 41
- 6.2 Suitability evaluation . . . 41

7 Conclusion **43**
- 7.1 Future work . . . 44

1 Introduction

The main paradigm of networks today have evolved around a *host centric* networking model which enables communication between two defined hosts. This communication model emerged with the first computer networks that was established in the 60's. It made it possible for a user to e.g connect and communicate with super computers at the time. With the introduction of the world wide webb in the middle of the 1990's, the communication model of information changed for the Internet, going from the *host centric* view towards a *data centric* view. With the Internet, it is the information produced by a website or a streaming service that is important, not necessarily the physical location of the content.

Information-centric networking (ICN) is a communication paradigm for the future Internet that is based on *named data* instead of *named hosts*. The communication is defined through requesting and providing named data, decoupling senders from receivers. This could make it possible to integrate storage for caching within the network infrastructure which could lead to improvements in the network as a whole [1] .

It is predicted that in the year of 2020 we will reach around 50 billion Internet of Things (IoT) devices [2]. These machines consists of high heterogenety and ranges from wireless sensors, e.g able to produce environment data, to wearables, smart home units and many other types of devices that was not previously connected to the Internet.

One area where these IoT devices differs from regular PCs, phones and similar, is that they have a different view regarding the usage of resources. For IoT units, things such as processing power, memory availability and power capabilities are most often very limited and those constraints are expected to stay within the domain in the future [3].

The usage of these IoT devices, and their data, often implies information centric usage pattern that is similar to the ICN [4]. Whereas, the content produced by the IoT device is consumed, by an user or application, on the network instead of communicating directly with the producer or host.

1.1 Problem statement

The purpose of this thesis project is to evaluate the performance and feasibility of using the ICN paradigm in the IoT domain. The scenarios are normal usage patterns between devices including producing and consuming data continuously created in a periodic interval.

A major goal of this thesis project is to do an in depth analysis regarding the performance and throughput of ICN used in the IoT domain. How long are the round trip times between a data consumer and producer, when using a ICN protocol, and are there any bottlenecks in the protocol as in processing-time or similar are questions that this thesis tries to answer. It is to be investigated whether or not ICN perform well enough in order to serve its consumers with data.

ICN is by nature a pull paradigm where a consumer has to initiate a request of a particular data object in order to retrive it. This is in constrast to several IoT systems where the publish/subscribe approach is more common. An example of such a system is MQTT [5] where the producer sends the newly created data towards a broker, which then pushes the data to a user/consumer. By evaluating the feasibility of ICN, a major goal is to investigate if it is possible to distribute data efficiently with ICN and a one time subscription model. It is in the interest of this thesis to

determine how stable such a system is and how well it perform over time. With the onetime subscription model, a consumer tries to pull the data from the producers by initiating a request just before the data has been created.

1.2 Delimitations

CCN will be the only ICN architecture covered in this thesis. Alternative implementations such as Psirp, Netinf among others, will not be covered at all. Furthermore, this thesis will not develop any further functionallity on the current CCN implementation for Contiki-OS. The current implementaion is to be considered sufficient. Exceptions are made for functionallity regarding monitoring and measurement metrics that will have effect on the evaluation.

Different cache strategies for the local storage at a CCN node and other functionallities that would be desirable, but not necessary, is considered out of the scope of this thesis. Furthermore, no aspect of power consumption will be taken into account for the thesis.

1.3 Research methodology

Currently there is a light-weight implementation of CCN available on several operating systems called CCN-lite [6]. In a former thesis project, Yanqiu Wu ported the CCN-lite implementation to the IoT operating system Contiki OS [7]. In this project, a qualitative and quantitative performance evaluation will be carried out through experimentation with this implementation. The sensor hardware used is the Texas Instruments SensorTag CC-2650, which runs the CCN-lite implementation.

Since the thesis is based on experiments a large amount of measurements will be conducted. The methodology used will be small loops between each measurement. In order to make the testing intervals short and feasible, a lot of effort has been put into creating smart scripts that automate testing and representation of the data so it becomes visual to the tester.

Different software tools will be programmed to measure the desired performance metrics. In order to evaluate the performance, various types of measuring tools will be conducted. These tools has either a general or a specific purpose. A general purpose tool can for instance collect the measured data from the test, and make it representable for the user. Specific purpose tools on the other hand can be ones that measure the processing time on a sensor or measure the latency time.

1.4 Contributions

This thesis contribute an experimental evaluation, with performance and feasibility aspects in focus, of using CCN on low power sensor hardware in a typical IoT enviroment where the data is sequentially produced in a periodic interval.

There are several other contributions of this thesis project. One is improving the current CCN-lite application for Contiki-OS, mostly to make it more stable and reliable when run on low power sensor hardware. Another contribution is the development of a tool, for the general CCN-lite project [**?**], which enables the software to calculate the round trip times for devices that runs the IPv6 protocol under the application level. Moreover, software enabling a consumer to retrieve data from the publisher through a one-time subscription model is described in more detail in later parts.

Furthermore, several testing tools have been developed to make this thesis possible. Although a lot of time and effort has been consumed into creating these software

implementations, the specific details concerning implementation will not be covered in this report. However, some higher abstractation regarding algorithms and logics will be presented.

1.5 Related work

When Jacobson et al. publicated the paper *Networking named content* in 2009 it sparked ideas of an alternative approach of communicating in contrast to current IP networking [8]. They implemented a prototype which replaced IP with CCN in the network stack and proved that it could be a potential alternative for the future Internet.

Since then, several research papers have been published comparing different ICN alternatives and their potential benefits and trade-offs when implemented as a network service[1]. Studies prove that it could be a suitable replacement of current IP networking structures, but there is a need for more performance analysis and studies[1][9].

However, it was not until the last couple of years that the research community started to investigate the feasibility and applicability of using ICN in the IoT domain. Several studies, the majority being literature studies or theoretic in nature, have been conducted recently or are currently ongoing. There only seem to be two implementation studies available to date.

In a conceptual study conducted by Ahlgren et al. [4], it is concluded that an advantage with ICN, is that the naming of data is independent of the device that produces it. The decoupling between a producer and consumer of data could improve performance in lossy networks. Challenges reside in the naming of data that is produced periodically over time where a major issue is to retrieve the *latest* value in that sequence. Potential solutions could be to implement a *one-time subscription*, where the request is stored in the cache at the node until the data becomes available[4].

Another study by Amadeo et al. [10] argues ICN is by nature close to the IoT domain. They also conclude that there is a need of further investigations regarding if ICN should be implemented as an overlay of existing IP infrastructure, coexist with IP, or if it should be a replacement in the same manner as proposed in [8].

Baccelli et al. were first to port of CCN-lite to the IoT operating system RIOT [11][12]. The project was the first trial of implementing ICN without any IP protocol in the IoT domain. They compared their CCN-lite implemententation and a regular 6LowPAN/IPv6/RPL approach and saw that there were several advantages using ICN over IP. Although they identified several areas where further work needs to be done, they argue that ICN is applicable in the IoT domain.

Another implementation of CCN for IoT devices was a thesis project conducted by Yanqui Wu at SICS/KTH[7]. He ported the CCN-lite functionallity into another IoT operating system, Contiki OS, and implemented the software as an middle-layer between the application- and the transport layer. Although some evaluation was done, there was no further investigation on how well CCN performs at the application layer.

1.6 Structure of the Report

This report is structured in six sections, omitting the introduction.

Section 2 will discuss background knowledge regarding the problem to this date. Technical details is presented that enable the reader to understand the details for the rest of the report. The background will first cover the network stack focusing on IoT

devices, thereafter ICN will be convered generally and CCN in more depth. A breif overview of the Contiki-OS gives the reader knowledge about the OS running on the sensors.

Section 3, implementation, discuss what components are targeted for the evaluation. A in depth description of the problem statement is discussed, aswell as identifying them. It is to provide an adequate evaluation of using ICN in the IoT domain, that certain parts of the CCN implementation is selected. These have been choosen with advice of my mentors, Bengt Ahlgren and Anders Lindgren. Which experiments the thesis perform, and why, will give the reader full knowledge to understand the evaluation sections described later on. Furthermore, an extensive setup section cover which tools were used to create the experiment enviroment used througout this project.

Section 4, latency performance evaluation, presents the methods used to enable the evaluation described in more depth in section 3. The results from the evaluation presents full insight in how well the CCN-lite application perform in the network. Parts of the results here, is to be viewed as necessary background knowledge in order to understand the upcoming feasibility evaluation in the next experiment.

Section 5, suitability evaluation, a method to following the data creation at the sensor is presented and implemented. The results provide argument to state that, with the correct software, CCN is a suitable replacement of MQTT or other publish/subscribe systems to retreive data from a IoT device.

In section 6 the results from the previous two sections are analyzed and discussed in order to reach the conclusion presented in section 7.

2 Background

By 2020 a total of 50 billion IoT units is estimated to be installed world wide. It represents a staggering 30 time increase from the year of 2009, where a total of 0.9 billion units was installed [2][13]. The increase in demand for IoT-related equipment and services has led to several major manufacturers joining the market, which in turn led to several different communcation standards which the IoT devices communicates with being developed.

A new communication protocol is not developed overnight. It takes a lot amount of research and time to develop a new protocol. A majority of the network protocols and systems in this thesis were developed during the 1990s or early 2000. Although the term ICN was coined in 2000 with the TRIAD paper [14], it was not until Jacobsson et al in 2009 [8], it started to draw wider attention.

Each operating system and protocol of communication has its own restrictions and possibilities in delivering sensor data from a producer to a consumer. In this section, the systems and protocols used in this thesis are described which can be applied in wireless sensor networks.

2.1 Internet of Things - network stack

IoT networks are designed after the traditional OSI model, where devices use the different protocol layers, as shown in figure 2, in order to communicate with each other. Due to the nature of IoT devices, where performance constraints and power limitations are in focus, some regular communication layer protocols are too heavy to be used within the IoT domain.

The wireless 802.11 (WiFi) standard is one such protocol where its high power consumption makes it infeasible to use for some IoT devices. With speeds over 500 mbit/s, it also provides a bandwith that is way above the needs of a constrained sensor device. There are several wireless alternatives, but the IEEE 802.15.4 standard is focus here because of its simplicity.

2.1.1 IEEE 802.15.4

The purpose with the IEEE 802.15.4 standard is to offer the fundamental lower network layers for wireless personal area networks (WPAN). The focus of the standard is on providing low cost, low power consumption and low data rates between inexpensive wireless devices. The standard only provides the MAC and PHY layers, leaving the upper layers to be chosen by the application [15] [16]. Due to the special PHY layer and to keep the transmission times short and resistant against failures, it does not exchange standard Ethernet frames with maximum transmission unit (MTU) of 1500 octets. The MTU of 802.15.4 is instead set to 127 octets and the maximum data transfer rate limited at 250kbit/s. Depending on wireless technology and how constrained the device is, the maximum transmission speed can be set to as low as 20 kbit/s.

There are two different types of network nodes that can exist in a 802.15.4 network [16]. Full function device (FFD) and reduced function device (RFD). A FFD node implements all communication funcionality the 802.15.4 standard offer. It can communicate with any other device in the network. A FFD node may therefore also route data from other nodes. When doing that the node is also called a coordinator.

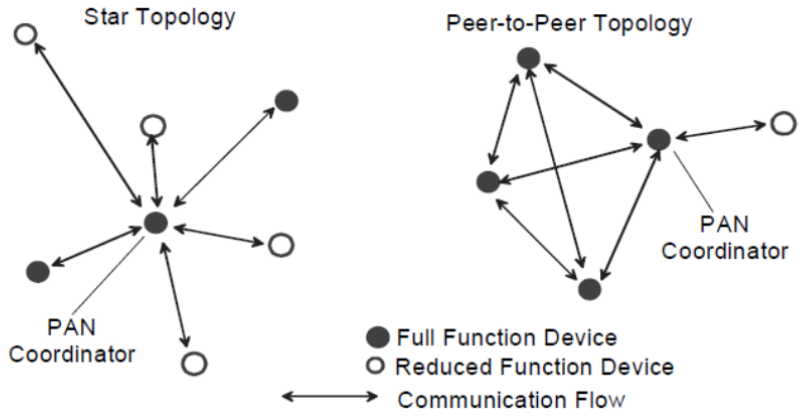

Figure 1: Topology structures in 802.15.4 networks. Star structure on the left, Peer-to-peer on the right.

If all communication in the network is routed through a dedicated FFD node, it is called a PAN coardinatior. A RFD has a reduced level of functionality and is meant to be extremly simple. Such devices are always an end node in a network and can only communicate through or with a FFD. They can never act as a cooardinator due to their limited capabilities.

The two main network topology forms that are used within the 802.15.4 standard, are the star topology and the peer-to-peer topology, shown in figure 1. In star topology, all devices are required to only communicate to a single central device called the PAN controller. An advantage with this topology is that it makes it easy to manage and support. The drawbacks of using a star topology structure are bigger, for instance will it limit the area that can be covered geographically since all data has to be routed through one device.

The peer-to-peer topology can have an arbitrary number of connections to each other within the network. Devices can communicate with each other, not only through the PAN controller, with exception for communication between RFDs. There are several advantages by using the peer-to-peer structure, for instance since devices can route traffic via other FFD devices, the network coverage can be easily increased. To be able to route traffic, the devices must have some ad-hoc routing protocol, for example RPL.

2.1.2 IPv4, IPv6

Since the introduction of Internet Protocol version 4 (IPv4) in 1981[17], it has been the backbone of the Internet and as the network layer in the OSI model. The protocol defines an address space of 32 bits and the total number of unique addresses that is available with IPv4 is around 4 billion. Today, the IPv4 address space is exhausting at a rapid speed and there is not enough addresses left to handle the increased number of devices that will be connected to the Internet in the future[Make citation!!].

In response to the shortage of address space, among other things, the Internet Protocol version 6 (IPv6) was formulised and defined as a successor to IPv4 in 1998[18]. The IPv6 protocol defines the length of an address to 128 bits, which lead to a total

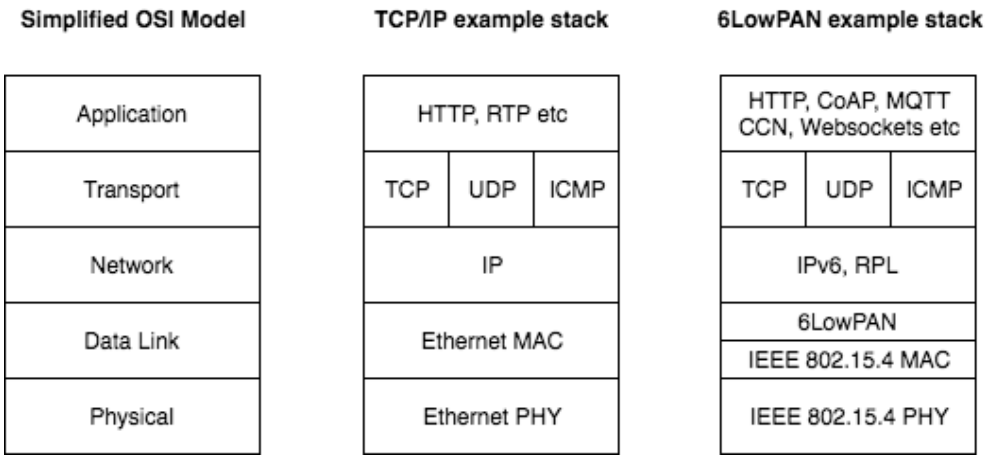

Figure 2: The simplified OSI model, an example TCP/IP stack and an illustration of an example 6LowPAN protocol stacks.

address space of 2^{128} equal to $3.4 * 10^{38}$ unique addresses. With an address space of this size, it will be sufficient for all IoT and Internet devices to have an own IP address. IPv6 requires that every link to the Internet has an MTU of 1280 octets or greater. In case this requirement can not be met, fragmentation and reassembly must be provided at layer below IPv6[18].

2.1.3 6LowPAN

At first glance, it may seem straightforward to send IPv6 data packets on a 802.15.4 network. However, there are incompabilities between the two formats making it hard for them to cooperate. For instance, the largest frame size of 802.15.4 (127 octets) is considerably less than the required MTU of IPv6 (1280 octets) [19]. Furthermore, the IPv6 header is 40 octets long which is almost a third of the total 802.15.4 MTU (of 127 octets). This leaves only 62 octets for upper-layer protocols as UDP or ICMP. That makes it impossible in the first case and infeasable in the second, to build IPv6 direclty on top of the 802.15.4 MAC layer as in a regular IP protocol on the Ethernet MAC layer in the IP stack, shown in the middle of figure 2.

To address these issues, among several, "IPv6 over Low-Power Wireless Personal Area Networks" (6LoWPAN) was established in 2007 as an adaptation layer between the IEEE 802.15.4 MAC-layer and IPv6. 6LowPAN makes it possible to transfer IPv6 packets over a 802.15.4 network through fragmentation and reassembly, and IPv6- and UDP header compressions to shrink the packet size. Through header compression strategies, it is possible to shrink down the IPv6- and UDP header, toward as little as 4 octets in total (instead of 48 octets) [19]. Other features of 6LowPAN is the neighbor discovery and mesh routing support. Even though there is no limitation to only use UDP, for simplicity and less overhead reasons, it can be more favorable to use UDP over TCP as the transport protocol with 6LowPAN.

2.1.4 MQTT, MQTT-SN

Message Queuing Telemetry Transport(MQTT) is a open lightweight publish-subscribe messageing protocol, designed for constrained devices with low-bandwidth and/or un-

reliable networks targeting Machine-to-Machine (M2M) communication[5]. The protocol reside in the application layer in the OSI model, assuming that the underlying network structure provides a point-to-point, session-oriented data transport provided by example TCP/IP [20]. This assumption makes the protocol unsuitable for devices that can not hold their own TCP/IP stack, which lead to MQTT-SN described further down. The publish-subscribe message pattern require a message broker, which is responsible for distributing messages to the subscribing clients.

The publish-subscribe pattern can be described by a server, or sensor, acting as a publisher/producer of information and a client as the consumer/subscriber of information. A client subscribes on a specific topic, set of data, that resides on the server/sensor. When the server has produced data for the specific topic, it will send that information towards the client. The information is going through a broker that handles all the information regarding which devices subscribe to which topic. The broker is usually located in a traditional network due to its higher performance requirements regarding bandwith and processing capabilities.

MQTT-SN, where the extension stands for sensor network, is a MQTT version that is adapted for wireless communication in constrained enviroments. It is optimized to be implemented on low-cost, battery-operated devices with limited or constrained storage and processing capabilities[21], particular targeting IoT and sensor devices.
Where MQTT uses string characters as topic names, MQTT-SN uses numeric IDs which reduce the size of the packets in favor of readability. Furthermore, MQTT-SN, in contrast to MQTT, does not depend on a connection-oriented transport service (TCP/IP), it is able to work with other transport protocols such as UDP/IP, ZigBee or others.

2.1.5 CoAP

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol to be used with constrained devices and networks, including M2M communication[22]. CoAP resides in the application layer, above transport layer, in the OSI model stack. It provides a client/server interaction model between application endpoints similar to the HTTP standard. CoAP is based on the REST architecture and follows the general design to manipulate data in a request/response manner. The methods GET, POST, PUT and DELETE are similar to HTTP, but not identical.

While HTTP uses TCP as transport protocol, CoAP data is sent asynchronously over a datagram-oriented protocol such as UDP. Due to the implementation of UDP, features like resend lost datapackets, and acknowledge-messages are missing in the transport layer[ref kid]. This functionality has in some extent been moved into the CoAP protocol and is called Messages.
CoAP defines four different type of messages: confirmable, non-confirmable, acknowledgment, reset. They occupy 2 bits out of 32 bits of the total CoAP header. A confirmable message provides reliability by retransmitting a message until a recipient sends an Acknowledgment message with the same Message ID back to the requester. If a requested message can not be handled by the recipient, a reset message will replied instead. When a message does not require reliable transmission (no acknowledgment is needed), a non-confirmable message is sent. These non-confirmable messages will still have a message ID in order to detect duplication.

2.2 Information Centric Networking

Information-Centric Networking (ICN) is a communication paradigm for the future Internet that is based on *named data* instead of *named hosts*. It represents an evolution of the Internet from todays host-centric networking, in particular IP networking, towards an information-centric approach.

There are several different approaches of implementing the ICN paradigm, but there are fundamental ideas that they all follow regardless which implementation is used. This section will continue describing the general ideas. However, the thesis overall will only consider the *Content Centric networking* approach described more in depth at the next section.

Some of the main features in the ICN are the possibility of in-network storage for caching data, multiparty communication through replication, decoupling senders and receivers, and that data is named[1]. In information-centric networks, an information request may not only be satisfied by locating the original information source, it is possible for any of the in-network caches to reply to the request with the desired data if they hold any copies of it. After a request is sent from a user, it is the responsibility of the network to locate the best source that can provide the information. Due to the fact that information/data is named, addressed and matched independently from its location, the data may be located anywhere in the network[icn research][18][19]. Hence the argument that the ICN approach decouple the information from its source.

2.2.1 Content-Centric Networking

The CCN communication [8] is driven by consumers of data. There are two types of message in CCN, *Interest* and *Data*. A consumer issues an *Interest* message to request an information object on the network. Any node that recieved the interest and containing the requested information, will respond by sending the *Data* back to the *Interest* issuer. A *Data* message is only sent in response to an *Interest* message, upon response, the interest message will cease to exist in the network. This implies that the data that a producer creates will only reach the network, or any participant of the network, if there is a request for it.

A key feature of CCN is that the content names are hierarchical. This allows routing information to be aggregated which in turn is critical in order to scale the network. The content name can be simila to URLs, for example a valid content name could be */sics.se/kista/floor/six/sensor/two/temperature*. However, there is neither a strict need for them to be human-readable nor tobe a URL. The prefix */sics.se/kista/floor/six/sensor/two/* could easily be exchanged to become either a hash value or just an integer, say *2* (representing the sensor two), whereby the same data could be accessable by the name */2/temperature*.

It is to be considered an interest match when any part of the interest name equals the named prefix of the data, for example is it possible that */2/temperature* could be match by */2/temperature/sequence_1*. If the data is produced periodically with sequence numbering, a consumer can 'follow' the data by the same manner once it has a starting point.

Even though there are a lot of similiarities to regular routers IP, there are a lot of differences between a CCN router and a regular IP router. Every Content Router (CR) maintain three main data structures: The Forwarding Information Base (FIB), the Pending Interest Table (PIT) and the Content Store (CS).

The FIB is used to map information to on which output interface a Interest message should be forwarded to in order to reach its content. It is very similar to a

regular FIB in a IP router, with the major difference that the CCN FIB allows lists of outgoing interfaces instead if just a single entry per object.

The PIT maintain a table for all incoming *interests* messages recieved by the CR, their current state and a mapping to which face they came from. When the data message for a particular *interest* arrives to a CR, the data message will be forwarded back on a reverse path, towards the face that exist in the corresponding PIT entry. After the data message has been forwarded towards the requester, its entry in the PIT will be erased. Whenever an entry is dropped or lost, for instance due to timeouts, it is up to the consumer to issue a new *interest* message.

The CS act as a local cache for information objects that has passed through the CR.

With the use of the CRs, CCN has great support for data caching. As stated earlier, once a *interest* is receievd, the CR will look through its CS in order to find matching data. Once the data is on the reverse path to the consumer, it will be put in the CS for a limited period of time for futhre use. Although there are several benefits using the cache in a distributed network system, it should be pointed out that this is not a long-term storage since a router can not hold infinite number of data. Nor is it useful for data objects that are requested at most once, since the benefits only occur when the data is requested a second time [4][1].

An example of CCN in action is illustraded in figure 3. Here, a consumer wants to retrive the indoor temperature data from the producer. Consumer1 sends an *interest* for data */temp/indoor* towards CR1. When it arrives, CR1 looks for data in its CS that matches the requested prefix of the interest. Since there is no match in the CS, the router performs a lookup on the longest prefix that matches its FIB in order to decide where to forward the incoming interest. When the match in the FIB is found, the router inserts the interest, with the incoming interface, into the PIT.

The same procedure happens for CR2 and the interest will be put in the PIT and forwarded to the producer. When the *interest* reaches the producer, it matches the name of the *data* and thereby the *interest* message is discarded and the *data* is sent back towards CR2. When the data is received, CR2 stores the data in the cache. Thereafter it performs a longest-prefix match in its PIT to get which interface it should respond to. In this case, it will respond to CR1 and the same forward back procedure will occur at CR1 until the data reaches the consumer.

When Consumer2 later on wants the same content, it sends an *interest* to CR1 and will retreive the data directly from its cache and thereby reducing the network traffic.

2.2.2 Using ICN for IoT

The usage of IoT devices most often implies an information centric pattern. In many scenarios, the main goal is the data and services and it is less important to communicate with a specific device [4]. Where users and/or devices rather consume content, generated by an IoT device, through the network than connecting directly to a specific device or host. Therefore one could argue that naming the data is more important than naming the devices.

Depending on topology structure of the IoT network, the caching mechanisms ICN provides could help constrained IoT devices to avoid unnecessary transmissions when distributing its data into multiple places. Storing cached data in the network could also potentially save energy and network bandwidth of an IoT device.

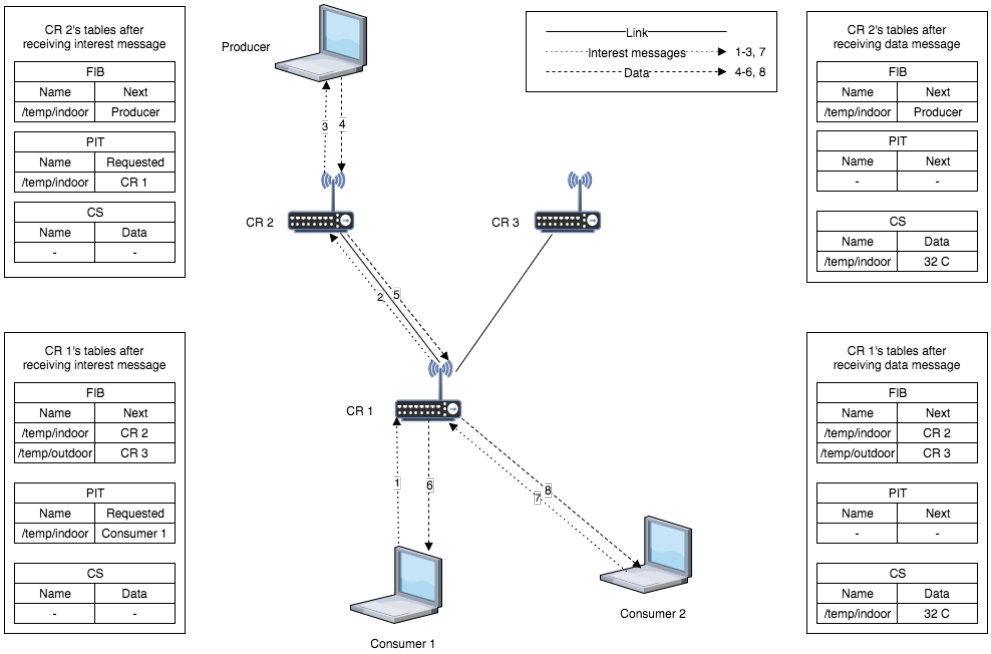

Figure 3: The CCN architecture builed up by content routers (CR), forwarding information base (FIB), pending interest table (PIT) and content store (CS), inspired by [ref survey ICN]

2.2.3 CCN-lite

CCN-lite is a lightweight, functionally implementation of CCN [6]. There are several platforms that are supported such as UNIX, Linux, Android, Arduino and several other. It uses a small code base of C, less than 2000 lines of code, in order to implement the CCN functionality. CCN-lite runs over UDP and Ethernet, and support packet fragmentation.

2.3 Contiki-OS

Contiki OS is an open source operating system that is suitable for network-connected, memory-constrained devices that targeting Internet of Things [23]. It focus on wireless technologies and implement a number of IoT protocols including 6LoWPAN, IEEE 802.15.4, RPL, CoAP and MQTT. Contiki provides a full IP network stack with protocols as IPv4, IPv6, TCP, UDP and HTTP. It is designed to operate with extreme low-power systems.

2.3.1 Sparrow

Sparrow is a commnunication format that encapsulates different types of payload on top of IPv6/UDP [24]. The Sparrow border router is based on the original Contiki border router but it has been improved with additional features. It acts as the RPL root and handles all the routing towards the sensors and maintains the network as a whole. The software makes it possible to initiate and hold communications with the remote sensortags. The border router connects the sensor network to the local host (Linux/OS-X or other), making it possible for the applications on the host to reach

nodes in the sensor network.

2.3.2 CCN-lite portation

Yanqui Wu, former thesis worker at SICS, implemented and ported a version of the CCN-lite [6] to the Contiki-OS platform in 2016 [7]. The portation enables a sensor to send and receive its application data with the ICN communication pattern providing a subset of the functionallity that the regular CCN-lite provides. A consumer can send an *interest* message to the sensor, running with Contiki software, in order to receieve the *data*. Depending on how much memory available at the sensor, one one can tune in how many entries that the PIT-table and the content store can contain.

3 Implementation

One percieved disadvantage with the ICN communication model, lays in the nature of ICN with its client-driven communication paradigm as described earlier. The data will only pass through the network if it has been requested. With a different approach, as in MQTT, the creator of a data can push it towards a user, which could potentially lead to low latency from data creation to when it is available at the consumer. This is impossible with the ICN paradigm. If the consumer does not know when the data is created at the producer, it could lead to remarkable longer latency time between data creation and its consumption compared to MQTT.

In order to achieve the same functionality, where the consumer retrieves the data from the sensor, an *interest* request must have been initiated. Together with a one time subscription approach ICN could give the same functionality, as in MQTT, when data is produced periodically over time. To make such a system feasible, the performance of ICN has to be good enough so the system is not a bottleneck to itself. The optimal case is if the consumer could retreive the data as soon as it has been produced.

Through experimentations, this thesis evaluates the performance and suitability described in earlier paragraph, of the ICN application CCN.

In the first experiment, in section 5, an evaluation regarding the latency performance is conducted. It covers differences in latency times between ping- and CCN peek requests between a consumer and a producer. The result provides arguments in favor of CCN, and they are later on used in the algorithm described in the second experiment.

The other experiment, covering an evaluation of the suitability using CCN can be found in section 6. It covers an analysis regarding the algortihm developed during the thesis and its performance from both a consumer and producers perspective. The algorithm tries to deliver the lowest possible latency, from the time the data has been produced until it is available to the consumer.

3.1 Experimental Setup

The implementation of the experiments described in previous paragraphs consists of a combination of hardware and software. The setup is shown in figure 4, where the whole system consists of a computer, Raspberry Pi 3, Zolertia Firefly and sensor. Together they communicate through various of protocols described both in section 2 and later on in this section. In order to provide logs and data for the experiments, the sensor is connected through USB to the computer. These logs can be read in real time or for later use, in order to verify the correctness of the system. All communication between producer, the sensor, and the consumer, a CCN application running on Raspberry Pi, is made through a Zolertia Firefly that is connected on a Raspberry Pi.

3.1.1 Hardware

The hardware setup used in this thesis consists of Texas Instrument CC2650 SensorTags, Zolertia Firefly and Raspberry Pi3. The sensortag produces sensor data that represent a IoT device. It communicates wirelessly with the Firefly-radio that is mounted on the Raspberry Pi3 which acts as a border gateway in this project.

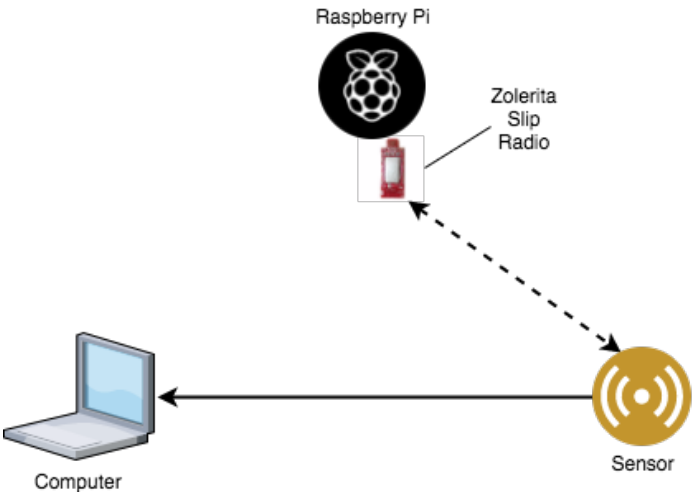

Figure 4: Overview of the experimental setup. The sensor communicates with the radio slip mounted on a Raspberry Pi. Through the slip, a consumer application on the Raspberry is available to issue interest request toward the sensor. The computer provides power to the sensor and in return it receives measurement logs.

3.1.1.1 Texas Instrument CC2650 SensorTag

The CC2650 sensortag is a wireless microcontroller developed by Texas Instruments [25]. The device is low cost, ultralow power device using the 2.4 Ghz radiofrequency to communicate with technologies such as 6LowPAN, Bluetooth and Zigbee. Due to its ultra low power consumption, the sensor can be powered by battery. The CC2650 device contains a 32-bit ARM Cortex-M3 processor running at 48 Mhz, accompanied by 8 KB of cache and 20 KB of SRAM. It contains a total of 128 KB of programable flash memory, which can be used for different application system such as the Contiki-OS system. The sensor controller supports the measurement of different types of sensor data such as temperature readings, optical light values and more.

3.1.1.2 Zolertia Firefly Slip radio

The Firefly radio slip is developed by Zolertia [26]. The radio slip provides a network infrastructure for the IoT devices, enabling them to communicate efficiently through the air. The Firefly has great routing capabilities due to its support for several communication technologies, among them IEEE 802.15.4/6LowPAN and Zigbee. Another advantage using the Firefly is that it supports multiple types of frequency bands such as 2.4 GHz, 915- and 920 MHz band. Radio parameters such as modulation, data rate and transmission power are highly configurable.

3.1.1.3 Raspberry Pi 3

The Raspberry Pi 3 is a single board computer developed by the Raspberry Pi Foundation [27]. It contains components suchs as WIFI, several USB ports, 1 GB RAM and a quad core ARM processor among several other features. Due to its relativly high performance for a low price, it has become a popular developing tool used in projects at home, in school and for academic research.

3.1.2 Software

The software setup used in this thesis consists of two CCN-lite applications [6] [7], Sparrow [24] and Ping6. The CCN-lite software is used on the Linux platform and the portation described in previous section is used together with Contiki OS on the sensor node. To make them communicate with each other the Sparrow software creates a gateway together with the slip radio on the Raspberry Pi. This enables a reliable wireless communication channel between the router and the sensor.

3.1.2.1 Gateway CCN application

The CCN-lite application used on the border gateway has the possibility to send *interest* message and recieve the *data* for the request it has sent. To issue an *interest* request, *the ccn-lite-peek* application is used. It is an utility tool within the CCN-lite software that can create, encapsulate and send out requests on the radiolink and also recieve the corresponding data packet. A CCN peek application runs only one time, a request either receives the data or times out after a given time, thereafter the application terminates.

Additional functionality has, in this thesis, been added to this software to make it possible to calculate the latency for the difference between an *interest* has been sent and when a response has been received. Throughout this thesis, CCN peek and peek is used interchangably in order to describe the round trip time from initiating a request to the time when that data has been received.

3.1.2.2 Sensor CCN application

The CCN application used for the sensors has a simple structure. Once the sensor has booted Contiki with CCN, it starts listening for incoming *interest*-requests from the network and produces content objects.

When an *interest* message is received at the sensor, a lookup in the cache will be performed to see if there is any matching content available. If there is a match in the *content storage*, then a data message will be created with the content and sent int response towards the issuer. Otherwise, the *interest* will be recorded in an entry in the *pending interest table* and there is no response toward the user until the requested data has been produced. An advantage is that this provides a possibility to ask for data that has not yet been produced. A consumer could potentially issue an *interest* message a short time before the *data* has been produced, which would firstly get the data directly to the user, and secondly reduce the latency on the network.

In every period a new content object is created. This object is cached into the *content storage* to be retreived later on by an *interest* request. There is also a lookup in the *pending interest table* to see if the newly created object already has an pending *interest* request. If so, the object will be sent out towards the issuer and the request is to be considered consumed. Once the storage is full, the content will be removed from the *content storage* in a FIFO-queue fashion.

3.1.2.3 Ping6

The command "live" tool Ping6 is a utility software for linux systems which use the ICMPv6 protocol to echo reqeuests as data over the network. In this thesis, it is used as a reference point for latency times in comparision to the CCN application.

3.1.3 Communication between gateway and sensor

When the border router communicate with the sensor, all technologies, hardware and software described in earlier sections come together and cooperate.

When a CCN peek request is issued, it starts at the CCN-lite application and becomes detected at the Sparrow border router software. The router will transmit the request using the slip radio toward the sensor. At the sensor, the request will be responded to or discarded. When the sensor responds with data, it will be forwarded on the reverse route backwards to the process on the router that initially made the CCN peek request.

All ping, CCN peek and data messages are sent through the 802.15.4 radio network. The sensor connects to the border router with the Sparrow software running on it. All the communication and message passing between the gatewey and the sensor node is made over the 802.15.4 network. Above the networking stack, the data is encapsulated into 6LowPAN packets, which contain a full IPv6 header (of size 40 byte). In Contiki OS, there are various header compression strategies for IP and UDP. In this thesis, all those compression techniques are turned off. Only the uncompressed 40 bytes IPv6 header is considered in this project. Thereafter, the application data is encapsulated by either UDP or ICMPv6 as their transport layer protocol. Both of those headers consists of 8 bytes.

3.1.4 Retrieve data

In all experiments, the sensor node is connected via USB to a computer in order to retrieve measurement values. The live command tool TTY captures all these messages from the console that the sensor produces and make them readable for a user.

After a few experiments, the result showed that the amount of information printed on the console had an impact on the latency. After several iterations, a decision was made that there would be two versions of printing. One printing only a minimum of information and the other one printing all the essential information the evaluation needed.

For verification purposes, the minimum printing sends information about whether an interest was responded to or not. For the essential printing, information regarding *interest* arrival times, *data* departure times and other metric values essential to the result were added. As a consequence, there is a higher consumption of processing power at the sensor. This is shown in later measurement results when minimum printing takes one tick on the clock (1/128th seconds) longer.

4 Latency performance evaluation

In this section a latency performance evaluation of CCN in the IoT will be conducted in order to see if CCN is any performance barrier. The result and thoughts from this section will lay as a foundation of the buildup and usage of the algorthim presented in the next section.

The purpose of this experiment is to measure and compare the roundtrip time, latency, between a sensornode and a border gateway using both ping and CCN peek commands. The results show which of these two alternatives has the lowest latency, how much they differ and if there is a common pattern between them. It is also interesting to see how much time it takes for a sensor node to consume an CCN interest, i.e receieve an interest message and respond it with the requested data.

During these measurements, the data is always available at the sensor for instant access, regardless if the ping or peek is used, there is no delay for the data to be produced.

From a more overview perspectrive, it is very important that the processing time of a CCN interest does not take to long time or to much resources and that it should be feasible for a sensor to deal with. If the computation time of returning data is too large, then CCN would be considered not suitable to be used for a IoT device.

From here on, latency and round trip time is used interchangeably aswell as CCN peek and peek.

4.1 Method

In order to have a reference point to relate to when the round trip time is measured, the ping6 command live tool is used which uses the ICMPv6 protocol to communicate between active devices. A similar tool with the ability to calculate and measure the round trip times for CCN-peek requests has been developed for the gateway CCN-lite application described in *section 3.1.2.1.* CCN-peek starts by issues an *interest* message for a data that is available at the sensor and stops when that data has been responded.

Both tools are used in the similar way as in figure 5, where the requestor/consumer is on the left-hand side and the sensor/producer is on the right-hand side. Time is represented on the Y-axis going from the top of the figure to the bottom. The latency is measured in time units from the requestors perspective, it starts when the interest/ping has been sent from the requestor and stops when the requested data has been replyed from the sensor. As figure 5 illustrates, the requested data that the ping command and interest message search for is always available directly at the sensor.

The ping6 command is used as a reference point in relation to CCN-peek because ping6 does not have any, or neglectable, processing time at the sensor. This fact enables us to calculate how long the processing time could be when using CCN-peek for latency measurements if all other components in the round trip time stays the same.

As seen in figure 5, one can translate the roundtrip-calculation for both ping6 and CCN-peek into the equation:

$$\textit{latency (roundtrip)} = \textit{interest transmission time} + \textit{data transmission time} + \textit{processing time} \tag{1}$$

$$\Leftrightarrow$$

$$\textit{latency (roundtrip)} = \textit{2 x transmission time} + \textit{processing time} \tag{2}$$

$$\Leftrightarrow$$

$$\textit{transmission time} = \textit{(roundtrip time - processing time) / 2} \tag{3}$$

where transmission time is the time it takes to transfer the data on the radio and processing time is the time it takes for the node to process the request. Queueing delay is possible in the system, such delay is here included under processing time.

In this experiment, the bourder router will perform 100 consecutive latency measurements towards the sensor using ping or peek. Thereafter the minimum, the median and average latency values are calculated from the result. The only variable that is varied in this experiment is the packet size of the outgoing data transmitted on the radio link towards the sensor.

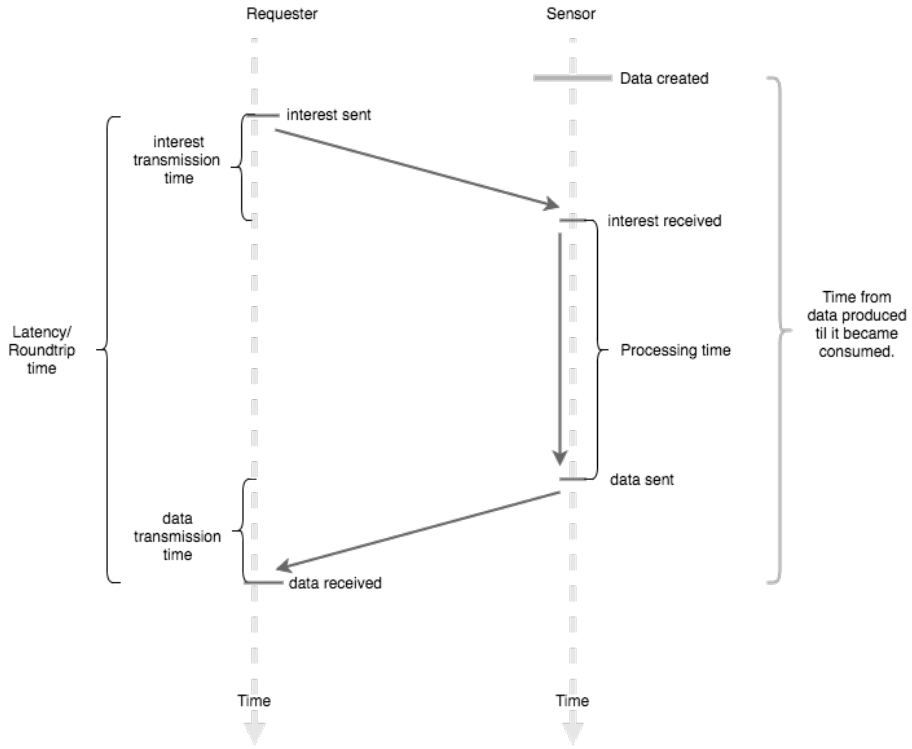

Figure 5: The roundtrip measurement is the time it takes for a client to initiate a request, and get data as a response. Transmission time is the time the bits spent on the wire or radio. Processing time is calculated from the time an interest was received til it was sent back to the requestor.

Ping and peek differs how the total packet size transmitted over the radio is choosen. With the Ping approach, one adds extra payload of data in the request by setting a flag and assigning how much extra payload the packet size should contain. For Peek on the other hand, in order to change the packet size one has to adjust the size of the data's name, i.e instead of sensor_temperature, it could be

sensor_livingroom_temperature, which would add 11 characters to the packet size transmitted on the radio. The shortest name a data can have is just one single character (which equals to one byte). In this experiment, the length of the named data will be of *1 + 5 * X*.

There is an underlaying assumption that the size of the outgoing packets from the wireless radio has to be the same in both cases ping and peek. The size of the interest packet and the data packet should also be of the same length. The reason the time spent on transmissioning the bits on the wireless link should be kept the same in both cases, is that otherwise it could affect the result giving shorter roundtrip times for one system. The latency is therefore the same in 1 and 2, where the transmission time is the same in both equations.

Ping and peek use the same network protocols up until 6LowPAN(IP layer), the header sizes of UDP and ICMPv6 are the same at 8 byte each. But CCN has an application header of 16 bytes and the shortest name possible of the named data is one byte. To make a ping message equal to the CCN header and the name, the experiment will use a ping payload of *17 + 5 * X*.

Table 1 shows the mapping of the size of named CCN peek data, the ping payload and how big the total packet size will be when it is transmitted on the 802.15.4 radio link. Row one shows the length of the named data in bytes, i.e the data with the name "sensor" would correspond to six bytes in size. In parenthethis, next to the size of the named data, one sees the total CCN peek application size, where the CCN header and the length of the named data is included. With the data named as "sensor", this would equal to 22 bytes in total.

The second row in table 1, shows the additional payload of a ping message, where the size of this payload has to be equvivialent to the total CCN-peek application showed in parenthethis one row above.

Row three shows the total amount of data that is transmitted on the 802.15.4 radio link towards the sensor. The total packet range in these experiments is an interval of five bytes from 93 up to 148 bytes. Even though 802.15.4 has a MTU limit at 127 bytes for sending data into one frame on the wire, it can be interesting to see how the latency varies when fragmentated as well. Therefore latency measurements are of packet size up to 148 bytes.

CCN Peek	1 (17)	6 (22)	11 (27)	16 (32)	21 (37)	26 (42)	31 (47)	36 (52)	41 (57)	46 (62)	51 (67)	56 (72)
Ping	17	22	27	32	37	42	47	52	57	62	67	72
802.15.4	93	98	103	108	113	118	123	128	133	138	143	148

Table 1: Row one shows the length of the datas name in bytes, if the data is named "sensor" it is equal to six bytes. The number in the parenthethis show the size of the CCN peek application, where the CCN header and the length of the named data is included. Row two shows the data payload of a ping message, which is equivialent to the CCN-peek application data. Row three shows the total amount of data sent on the 802.15.4 radio toward the sensor node.

4.2 Results

The results from the latency times with ping6 are shown in figure 6, where packet size, in bytes, sent on the radio link is shown on the x-axis and the latency time, in milliseconds, on the y-axis. The same axis layout holds for figure 7 and 8, but those figures shows the roundtrip time for CCN peek in two different cases. One is with the essential information printed out on the debugging console. The other one does not print anything on the debugging console.

The results, illustrade in figure 6, show that the median latency for a ping6 request is very close to around 25 ms from 93 bytes in packet size up to 123 bytes. The median latency when sending a CCN peek request without debugging, shown in figure 7 is little above 25 ms for packet sizes the fragmentation limit. Roundtrip times when debugging is turned on, illustrated in figure 8, show a median of around 36 ms. The 10-15 ms difference (around 40%) between the two peek results shows that there is a lot that can slow down the processing power, in this particular case it is only due to printing out information towards the user in the terminal.

When the experiments started, the measurements showed that the round trip times was around 130 ms in median values. Those latencies was considered very high and unrealistic. It resulted in an investigation regarding where the processing time was spent on the sensor. Several timing checkpoints was set out and it showed later that there was code that put the sensor in sleep mode for 100 ms every time the sensor retrieved an *interest*. It also revield several flaws in the CCN-lite portation that needed to be handled. When they were fixed, the result was a more optimal code base and better performing latency values which is shown in the previous paragraph.

The CCN peek (from here on only without debugging) latency time is only a few milliseconds apart from the ping counterpart when the same amount of data is sent of the network. This indicates that, in a bigger perspective, the time it takes to process an incoming CCN interest is almost negligible for the sensor node although it is constrained. Although the difference is very small, when comparing latencies in figure 6 and figure 7 one can see the small jump between them. This makes sense when considering that peek has to look up the content, process it and then respond to the requester, whereas ping would more or less respond direclty.
The time resolution on the sensor equals to 1/128th second, equvivalent to 7.8 ms. Unforturnately, this makes it impossible to further investigate in the details of where the processing time is spent on the sensor. The couple of milliseconds in difference between a CCN peek and ping is not enough for the time resolution.

In all tests, the latency remain relativly flat even though the packet size is increasing, except the region around 123 byte to 128 byte. This indicates that the transmission delay has small overall effect on the latency. It also corresponds well to the fact that it theoretically takes 0.5 ms to send 127 bytes on a link that has a transmission rate of 250 kbit/s.

The fifteen to twenty millisecond jump in latency from 123 byte to 128 byte can be seen in all tests and is due to the 127 byte MTU of 802.15.4 which result in packet fragmentation at 127 bytes. The latency remains stable even after the fragmentation limit, which makes sense considering the flatness of the latency described above.

All measurements show a five to ten milliseconds difference between the minimium

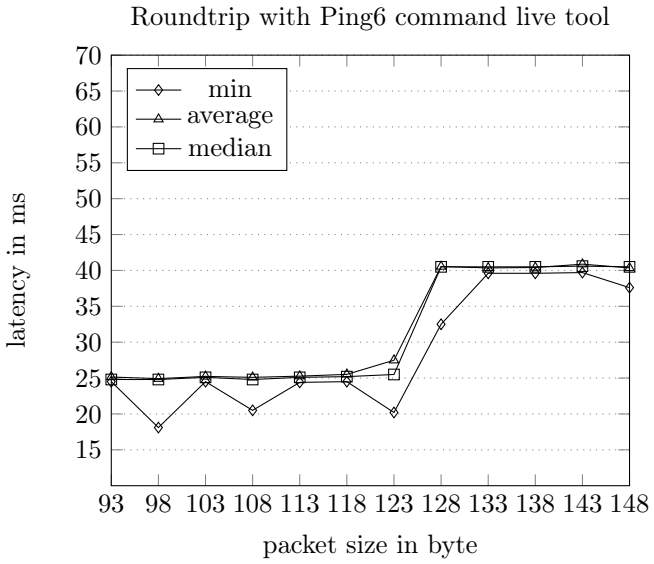

Figure 6: Latencies in milliseconds when pinging a sensornode with packet sizes from 93 byte to 148 byte transmitted over the radio. The latencies stays stable at around 27 ms up until 123 byte. After fragmentataion, the latency raise and stay stable at around 43 ms.

latency and the average/median latency. The histogram for the ping and peek latency measurements are illustrated in figure 9 and 10. They show the number of roundtrips, for application sizes of 17, 22 and 27 bytes, that can be categorised together and thereby see if there is any outliers that can be obmitted. Both histograms show that there is only a few such outliers and the absolute majority of the roundtrips lay around 24-28 ms. This inidicates that, even though the average and median latencies are close to each other, the median value is the correct way of measurement.

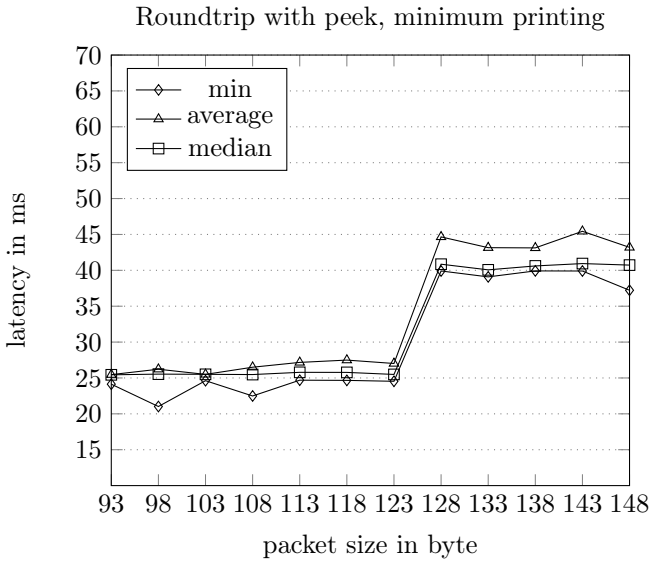

Figure 7: Latencies in milliseconds for peek interest requests of sizes from 93 byte to 148 byte. The latencies stays stable at around 26 ms between 93 byte to 123 byte. After fragmentation, the latency raise and stays stable at around 41 ms.

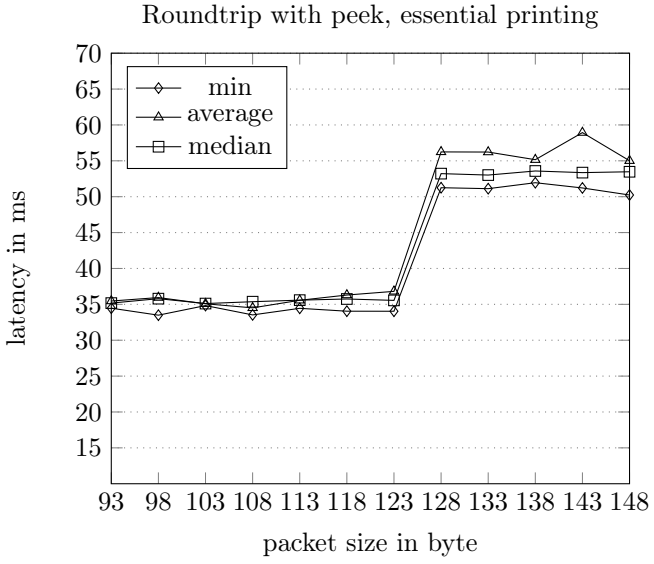

Figure 8: Latencies in milliseconds for peek interest requests of sizes from 93 byte to 148 byte. The latencies is stable at around 35 ms between 93 byte to 123 byte. After fragmentation, the latency raise and stays stable at around 53 ms.

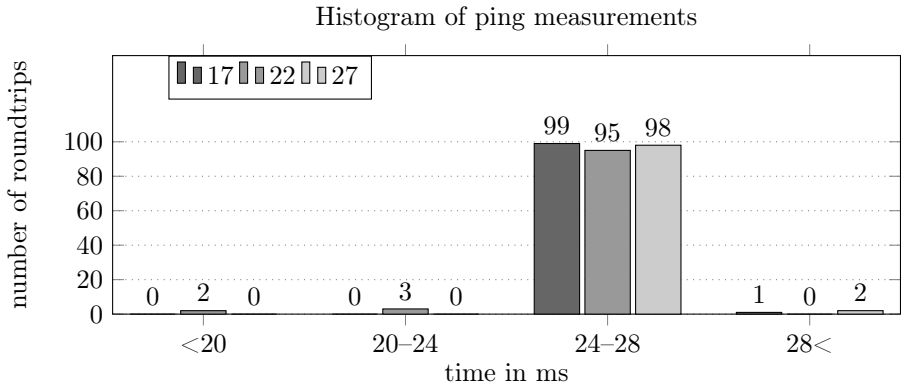

Figure 9: Histogram of ping latencies result. A majority of the roundtrips end up in the 24-28 ms span.

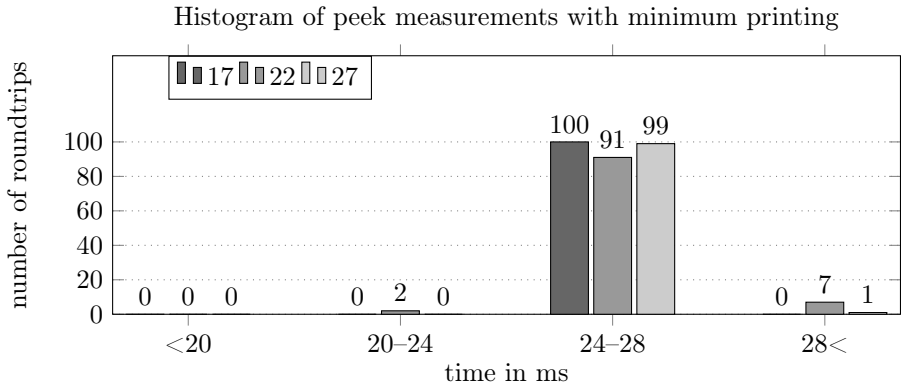

Figure 10: Histogram of CCN-peek latencies result. Almost all of the roundtrips ended up in the 24-28 ms interval.

5 Suitability evaluation

In the previous experiments concerning the performance evaluation, the consumer would always receive the lowest possible latency for data that was already produced and available at the sensor which can be seen in 5. Furthermore, one can also see that there can be a uncertain period of delay between the time of data creation until it reaches the consumer. As a user, one always want to receive the data as soon as it has been produced and the objective here is to reduce, or remove, the latency from data creation at the sensor to the time it reaches the consumer.

Communication protocols such as MQTT has a low latency between data creation and consumption by nature because the producer pushes the data towards the consumer when the data has been created. In ICN however, it is not possible by nature to push the data towards the consumer in the same manner as with MQTT.

With the possibility of storing *interest* messages at the sensors, combined with the 'one time subscription' approach, described by Ahlgren et al in [4] where the consumer sends an *interest* message in advance for future data, it could be possible to achive the similar low latency with ICN as with MQTTs push mechanism. A visual representation of this is illustrated in figure 11, where a user sends an interest message for the data named "s1" before it has been created at the sensor. Thereby subscribing for the data s1 and making use of the sensors pit in order to acheive low latency between data creation and retrieval of data. The result shows how such a system perfoms over time and if it is a suitable alternative to the publish subscribe approach.

The purpose of this experiment is to show that the latency between data creation and retrieval at the consumer does not have to be affected when using ICN. A key goal is to enable the consumer to receive the data as soon as it has been produced at the sensor. The rate when the data is produced could be of arbitray, but fixed, periodic length.

Although the purpose is to follow the sequence numbering of the data, it is out of the scope of this thesis to find the latest sequence number. This is considered to be known for the system in advance.

The expression 'data from the *PIT*' is equvivalent to once the data has been created, the sensor looks in the *PIT* for a corresponding *interest* message to consume and thereafter send the data towards the requestor. When the data is not from the *PIT*, it comes from the cache.

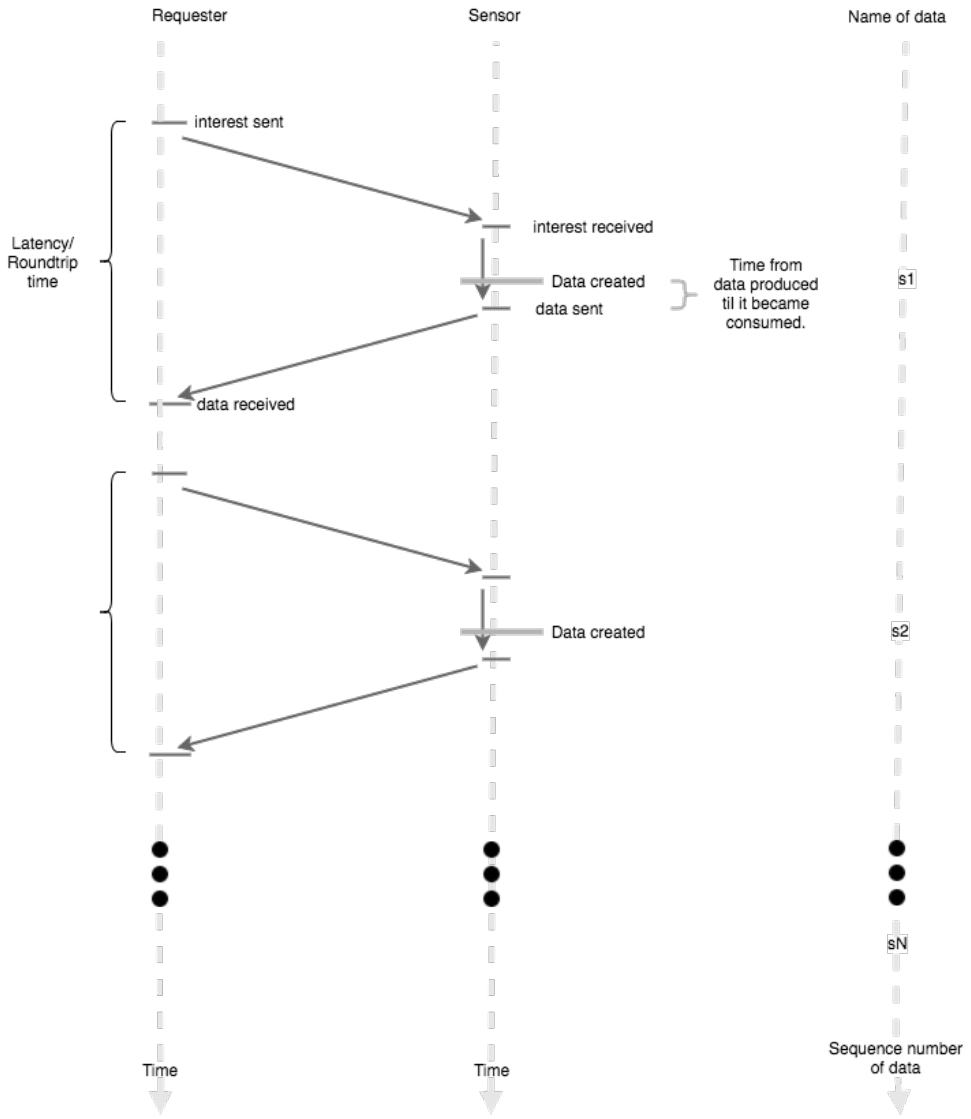

Figure 11: A users sends an interest message for data "s1" towards the sensor in advance in order to acheive low latency from data creation to receival. The idea is that this works for sequentially produced data over longer periods of time.

5.1 Method

In order for a consumer to receive sequentially produced data as soon as it has been created on a sensor, an algortihm has been developed in this thesis project. The algortihm, shown in figure 12, is a first attempt to retreive data with the previously described one time subscription approach together with CCN. It is based on two stages. In the first stage the consumer tries to go from receiving the data from the sensors cache to receive it by using the sensors *PIT* instead. In the second stage, the algorithm tries to fetch data at a certain interval in order to receive the data as soon as it has been created on the sensor.

The process of retreiving data from the sensors cache to using its *PIT* is done

by sending the next *interest* message in advance. To send it earlier, a factor of the interval is subtracted at each interval. This procedure continues until a timeout has been reached. Then the algorithm sends another issue of the same *interest* message. If the algorithm receives a response to the second message, it assumes that it now has started to make use of the sensors *PIT* and thereby sets the reference time to the time of the timeout. During this phase, the minimum latency value achived is saved together with the reference time to be used later on in stage two.

In the second stage of the algortihm, it tries to fetch data at a certain interval in order to receive the data as soon as it has been created on the sensor. As seen in figure 12, it calculates when to send the next request towards the sensor based on the round trip time of the current interval.

The smoothed round trip time (*srtt*) technique uses the Exponentially Weighted Moving Average (EWMA) to make the system more tolerant when handeling a latency value that differs substantially from usual. One must set a round trip target (*rtt_ target*), line X, that the algortihm follows, which sets how "early" the client should send the *interest* message. The higher the *rtt_ target* is set, the larger the rtt on line X will become. *rtt_ target* is a foundation of the algorithm and it is a factor of the minimum round trip time (*rtt_ min*). In the algortihm there is a background correction factor that is calculated at every interval. This correction time, line X, is the difference between the srtt and the *rtt_ target*. It represents the difference in time between the two systems. Together with the interval rate at which the data is produced and when the current interval started, the correction factor is added in order to decide when to send the next request on line X. At last it is calculated on how long the algortihm should wait until sending that next message on line X.

The reason for using the *rtt_ target* is the need to send the *interest* request at a comfortable distance before the data has been created. When using the *PIT* of CCN, one can not retrieve the same latency values as in the previous measurement where the data was replied directly from the sensors cache. Instead, one must set a round trip target that is related to the rtt_min. The difference between *rtt_ target* and *rtt_ min* defines how long the time span which the algorithm can regulate is. The smaller the differences, the more vulnerable the system becomes for variations of rtt. The minimum difference must be greater than the differences of the two intervals between the sensor and the gateway. If it gets less, the algortihm stops working properly.

In order to calculate when to send the next request towards the sensor, one could subtract the rtt from the interval. However, this makes the system more sensitive to changes in the latencies which could lead to timeouts for a consumer. To solve this issue and make the latencies more stable over time, a smoothed rtt is used in the algortihm. It uses a factor α, to decide how the current round trip time should be weighted in comparison to the previous latency values. The greater α becomes, the less importance the older values have. When α becomes smaller, the weighting of older values grows. If the latencies of the srtt is high, the *interest* request is sent too early towards the sensor. At the same time, if the srtt latency becomes to short, the request is sent to late.

```
1  rtt_min;
2  reference_time;
3  while True do
4  |  time_start;
5  |  rtt = send_interest_receive_data;
6  |  if Timeout then
7  |  |  second_rtt = send_interest_receive_data;
8  |  |  if Timeout then
9  |  |  |  reference_time = time_start + interval_time;
10 |  |  |  Break;
11 |  |  end
12 |  end
13 |  if rtt < rtt_min then
14 |  |  rtt_min = rtt;
15 |  end
16 |  time_end;
17 |  next_time = time_start - time_end + interval_time +
   |   interval_shorting_factor;
18 |  sleep(next_time);
19 end
20 next = reference_time;
21 rtt_target = rtt_min * x;
22 α = 0 <= α <= 1;
23 while True do
24 |  rtt = send_interest_receive_data;
25 |  if not timeout then
26 |  |  srtt = α × rtt + (α - 1) × srtt;
27 |  |  corr = srtt - rtt_target;
28 |  end
29 |  next_time = next_time + interval_time + corr;
30 |  sleep(next_time - current_time);
31 end
```

Figure 12: Algorithm that makes it possible for a consumer to follow the creation of data with a certain interval. The first stage, line 1 - 19, receives data from the sensors cache. In the second stage, line 20 - 31, the consumer receives data from the sensors PIT.

A mockup testrun of the algorithm can be seen in figure 13 and 14, where 13 shows data from a sensors/producer perspective and figure 14 shows it from the gateway/consumer perspective. The age of data is illustraded on the Y-axis in figure 13, which represents the time when the data was created in relation to the time when an interest was received of that data. Formalized, the Y-axis is equal to the equation:

$$\textit{Age of data} = \textit{Time data generated - time interest received} \tag{4}$$

If the interest message can be responded directly after it has been receieved at the sensor, then the response is coming from the sensors cache and the age is positive on the y-axis. When one uses the sensors PIT, the age values on the Y-axis becomes negative and the sensor receives an interest message before the creation of the data. This makes it possible to achive low latency times from data production until it reaches the consumer. The round trip times from a gateways perspective is shown on the Y-axis in figure 14. In both figure 13 and 14, the X-axis represents a time line where data is generated periodically and named sequentially with a fixed rate that is set before-hand. If nothing else is mentioned, the rate is one data value generated every second.

The effect of the first stage of the algortihm can be seen in on the left-hand side of figure 13 where the age of the data is successively decreasing because the *interest* message has come in to the sensor earlier compared to the earlier message. From figure 14, one can see that the round trip times stays stable during the initiating phase. Note that both these figures contain mockup data for illustration purposes.

The age of the data at the sensor shown in figure 13 are under zero when the *PIT* is used. It shows the time it took from when the *interest* message was received at the sensor until the data was created and sent back towards the issuer. A negative age implies that the sensor data was created after the *interest* message was received. The time it takes from data creation to the time it is sent out from the sensor is almost zero, or just the processing time, which can be deduced from figure 5 in the performance evaluation.

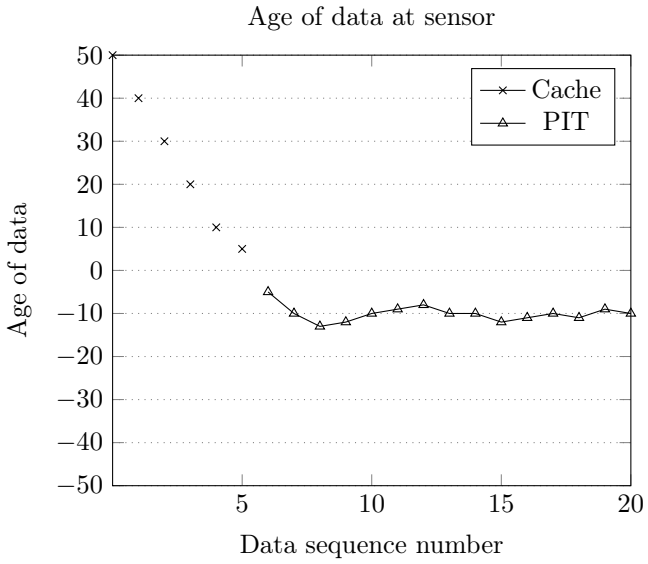

Figure 13: The time it takes for an interest message to be consumed in relation to the creation of the data. When the age is positive the data comes from the sensors cache otherwise it comes from the PIT. The mockup data in this graph is for illustration purposes.

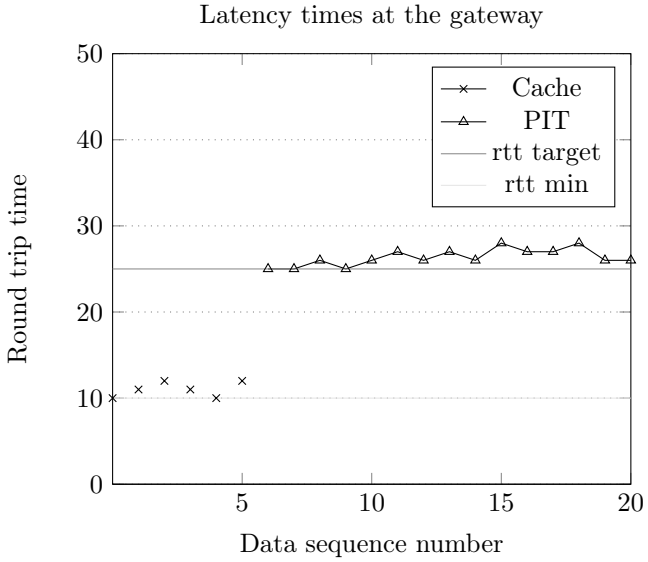

Figure 14: Round trip times when using the algorithm. Up until the fifth interval, the algorithm fetches data and calculates the lowest latency by retreiving the data from the sensor's cache. When it starts to use the PIT, the algortihm tries to acheive round trip times close to the rtt target which is a factor of the calculated rtt min. The mockup data in this graph is for illustration purposes.

5.2 Result

The goal of these experiments is to achieve low latency for data that is sequentially produced until it is sent out towards their consumer. In the experiments, the Sensor CCN application, described in 3.1.2.2, is used to produce data. To consume data, the algorithm described and developed in previous paragraphs uses the Gateway CCN application to issue *interest* messages and receive data that is produced by the sensor application. The communication flow between the two applications is described in chapter 3.1.3.

The results regarding latencies where the impact of the current rtt is low, $\alpha = 0.1$, and the algortihm sends the *interest* messages long time in advance, $rtt\_target = 2.5$, is shown in figure 15. The interval sequence number is plotted on the x-axis and the roundtrip latency is plotted on the y-axis. The same axis layout holds for all figures of which regard the round trip times from the gateway/consumer perspective. The results, illustrated in figure 15, shows that the latency is stable between 90-100 ms, the srtt is stable with a small variance around 95 ms and that the correction is stable around 5 ms as well. This results indicate that the consumer can retrieve the data in a stable manner.

The corresponding time from a sensors perspective is shown in figure 16, where the interval sequence number is plotted on the x-axis and the age of the data is plotted on the y-axis. When the values become negative, the sensor makes use of the CCN *PIT*. They show the time between recieving the *interest* until the *data* was created and responded towards the requestor. The same axis format holds for all figures regarding the age of the data at the sensor. Figure 16, which corresponds to when $\alpha = 0.1$ and $rtt\_target = 2.5$, shows that the age of the *interest* is very stable at 62 ms (8 tick).
Small variances in the age of the interest indicates that the algorithm to request the values is stable and that the use of the pit is working properly.

In order to give the current rtt value greater significance compared to older values in the algorithm, the α parameter is changed to a higher value. When it is set to $\alpha = 0.9$, the resulting curves show a more alternating from compared to $\alpha = 0.1$ as illustrated in figure 17 and 18. The latencies range from 87 ms to 117 ms, and most often they are either at the both ends spectrum or at 97 ms. The age of the *interests* is at the same time varying between 56 ms to 70 ms. This is due to the fact that the srtt has less importance and influence as α grows. Instead, the time when next interest is sent is more depending on the current latency time. These results show that the algorithm has no problem handling a shifting rtt and that the system stays stable. When the srtt start to alternate, it is hard for the system to stop.

When α is set to 1, there is no smoothing, therefore the correction is only dependent on the last roundtrip time. The result with no smoothing is illustrated in figure 19 and 20, and it shows that the latency times alternate between 87 ms and 117 ms as seen with $\alpha = 0.9$, but here the frequency is greater. The algorithm works and can easily handle when changing α to 0.9 or 1, but this is also due to the high rtt_target. If the gap between rtt_target and rtt_min shrinks, the risk of instability and retrieving older data grows.

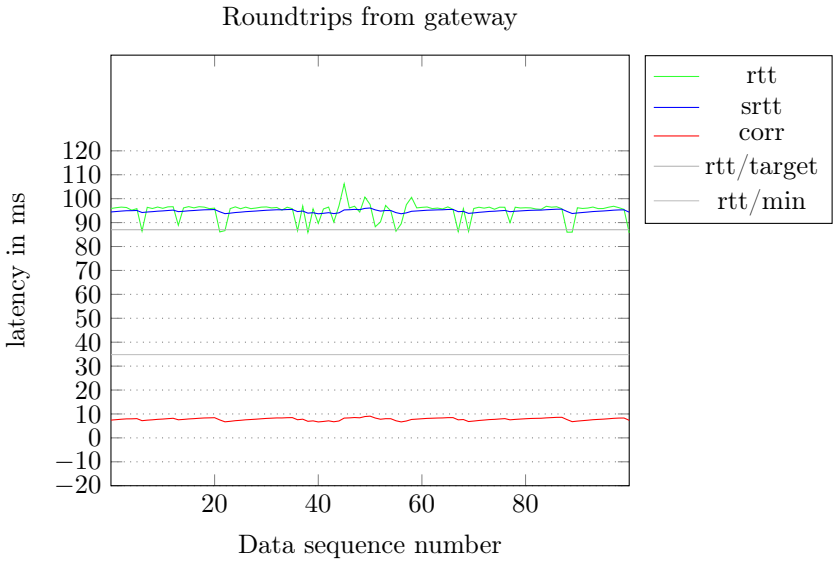

Figure 15: Latency values from the gateway when α=0.1, rtt_target=2.5 and there is no artifical difference in interval length between the gateway and sensor clocks. The result shows that when α is low, there are small variances of the srtt and the correction factor which gives the system these smooth values.

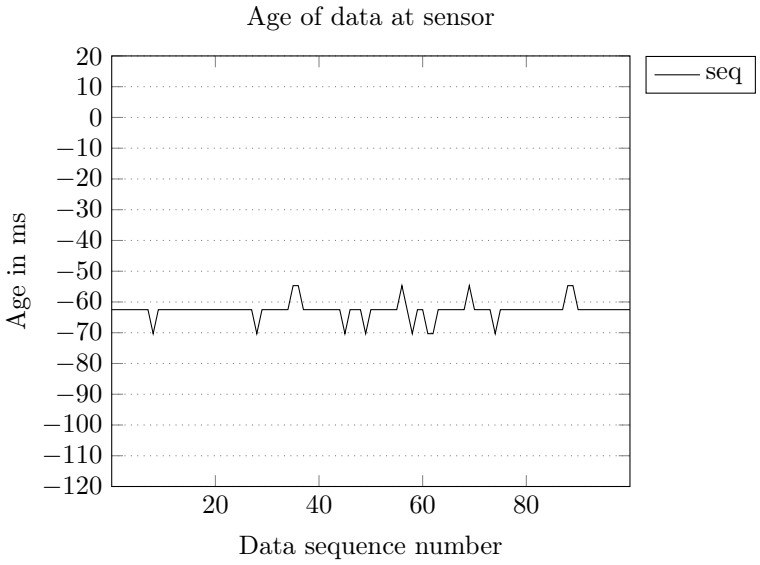

Figure 16: Age of the data when α=0.1, rtt_target=2.5 and there is no artifical difference in interval length between the gateway and sensor clocks. There are small variations in the age of the data, which is expected considering the latencies from 15.

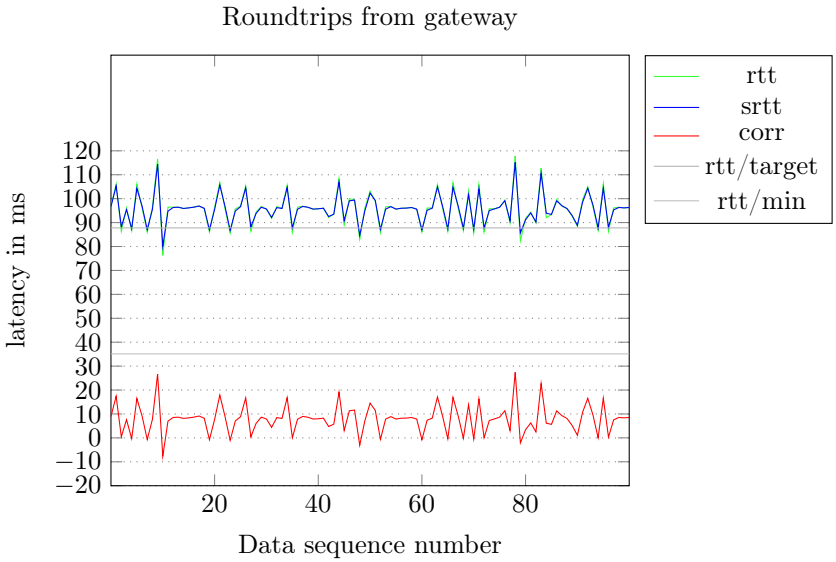

Figure 17: Latency values from the gateway when α=0.9, rtt_target=2.5 and there is no artifical difference in interval length between the gateway and sensor clocks. The result shows that when α is higher, the variances of the srtt and the correction factor grows in frequency. The algortihm has still no issues to function normally.

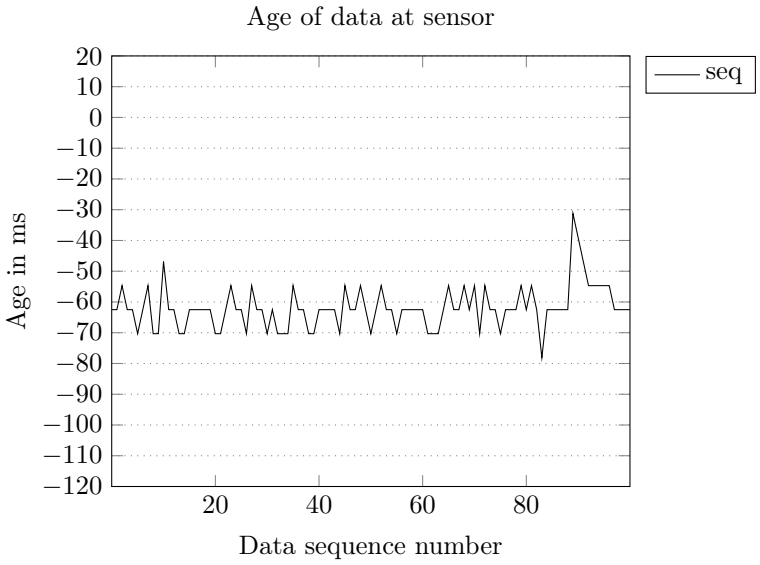

Figure 18: Age of the data when α=0.9, rtt_target=2.5 and there is no artifical difference in interval length between the gateway and sensor clocks. The frequency of the variations of the aged data increases because of the higher srtt described in figure 17.

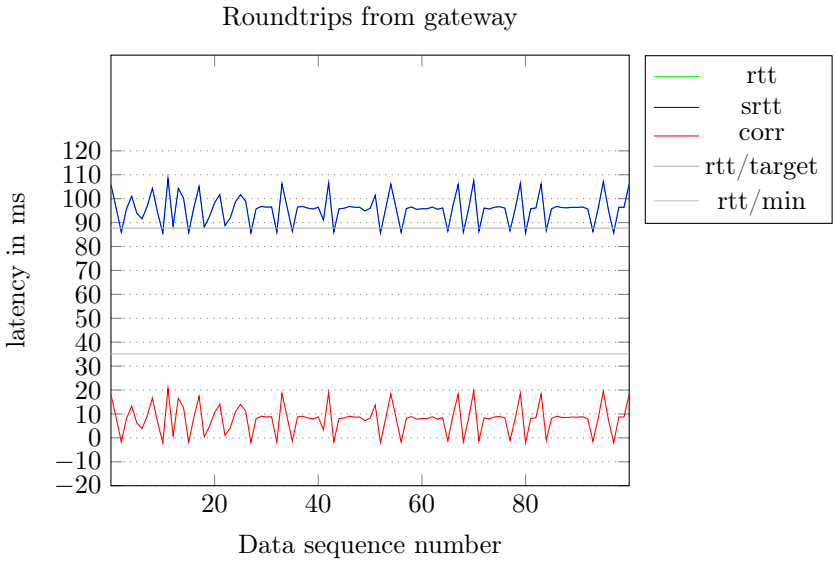

Figure 19: Latency values from the gateway when α=1 rtt_ target=2.5 and there is no artifical difference in interval length between the gateway and sensor clocks. Although the system stays intact when not using the srtt, the frequencies of the variations of srtt and the correlation factor continues to grow. Yet, the main functionality of retreiving data without issues is untouched.

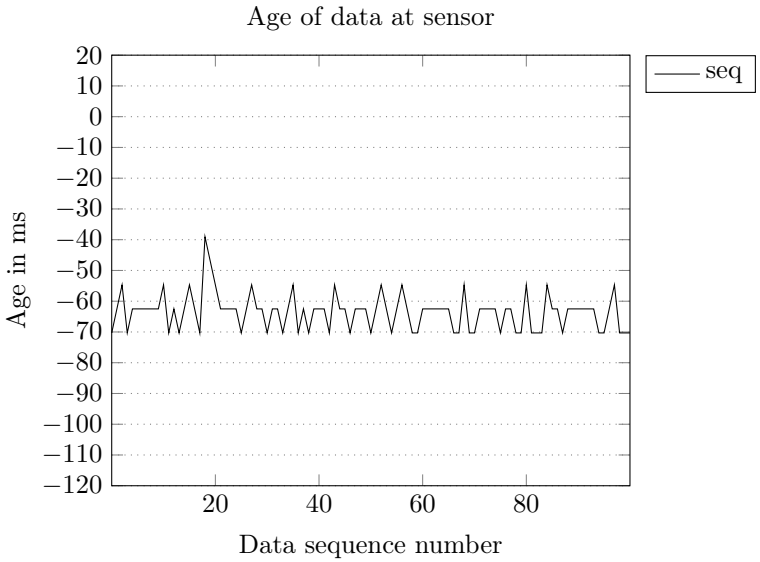

Figure 20: Age of the data when α=1.0, rtt_ target=2.5 and there is no artifical difference in interval length between the gateway and sensor clocks. The frequency of the variations of the aged data is high because of the high srtt and correction factor described in figure 19.

5.2.1 Different intervals at the sensor

All of the previous results were made under the assumption that the clock of the sensor and the clock of the client and their speeds are relative the same. One interval second on the sensor is almost the same as one interval second on the GW. To illustrate the functionality when the two clocks are not synchronized, and thereby run at different speeds, other test have been made. The algortihm proves to stay stable even when the sensor is providing data at a different periodicity than the gateway.

In figure 21 and 22, the sensor is creating the data with an interval of one second + 2% and the gateway still uses an interval period of one second. The results show that the system is still stable over time, even though here the interval is larger, simulating a slower clock on the sensor. As figure 22 shows, the time it takes from the *interest* message arrives at the sensor to the time the data has been created and responded back to the gateway is around 87 ms. The latency from the gw is overall higher, 105 ms to 125 ms, in comparission to without any difference in the time intervals, but the variation of srtt is still small and at the same levels.

This is logical since the *interest* is sent earlier from the gateway towards the sensor, but at the the same intervals, due to the algorithm. However, there is a cap, when the *interests* start to timeout, the system will stop working properly and no data will be received at the gw. The drift on the sensor compared to the gateway cannot be greater than the timeout that the gw is setting on the *interest*-packet when it is issued. Furthermore, the srtt is always greater than rtt_target which provides stability in the system.

Additionally, at a smaller sensor period of -2% compared to the gateway-interval, the system stays stable. The roundtrips as well as the age of the interests are smaller than when the two intervals are the same and the age of the *interests* are smaller too. The majority of latencies is between 65 to 75 ms, as seen in figure 23, and the time it took to consume the interest was around 40 ms as seen in figure 24. The reason why this is still stable is because the difference between rtt_target and rtt_min, 88 ms - 38 ms = 50 ms, is greater than the negative drift of -2 %, approximatly 23 ms, of the sensors interval clock. Once the drift exceeds the difference between rtt_min and rtt_target, the algortihm is unstable and it stops working properly.

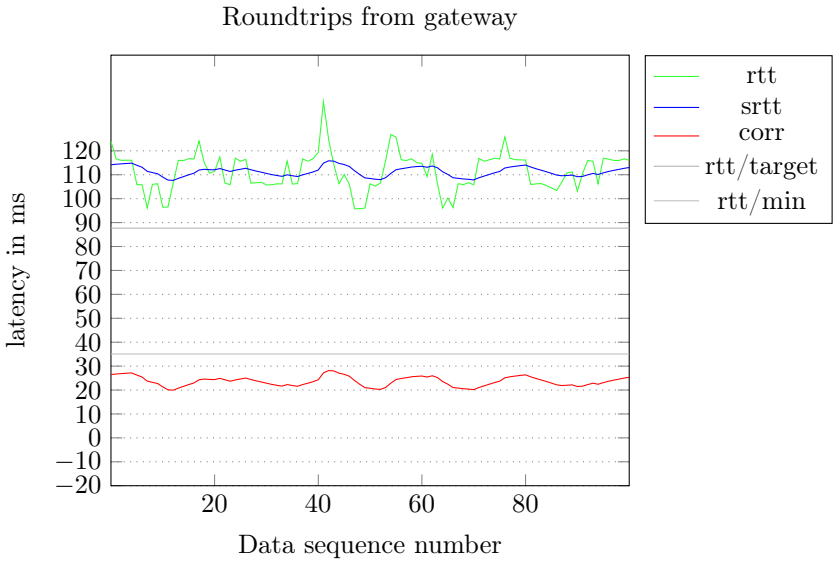

Figure 21: Latency values from the gateway when α=0.1 rtt_ target=2.5 and there is an artifical drift in length of the sensors interval of +2% compared to the gateway. The algortihm has no issues to handle a positiv drift in time, where the intervals between producing content is longer on the sensor compared to the gateway. The algortihm handles this drift by sending its interest messages earlier towards the sensor, resulting in higher srtt times overall.

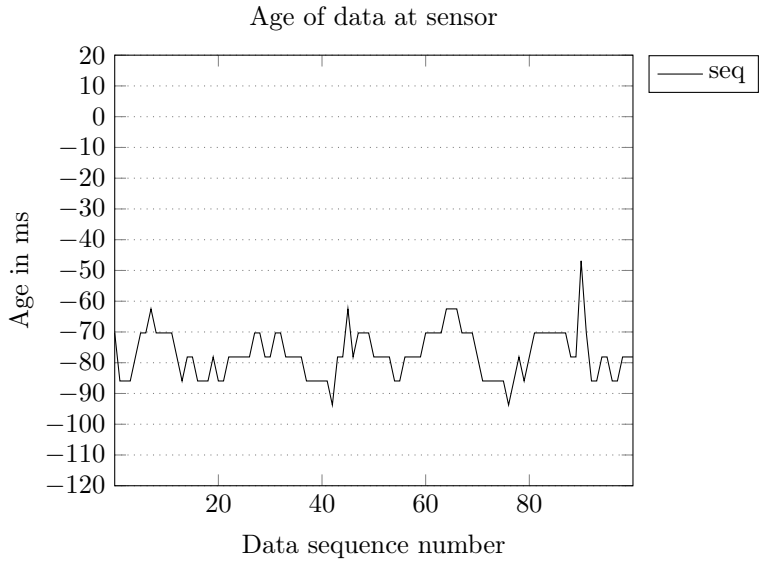

Figure 22: Age of the data when α=0.1, rtt_ target=2.5 and there is an artifical drift in length of the sensors interval of +2% compared to the gateway. One can see that the age of the grew compared when there was no drift. This is because of the algortihm is sending the interest messages earlier.

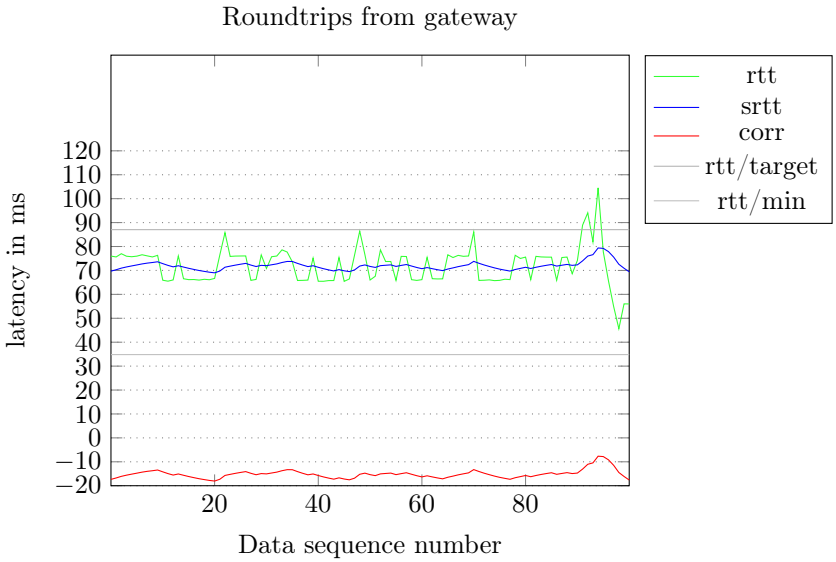

Figure 23: gw/test/alpha/0.1/rtttarget/2.5/diff/0.98/slope/0.15/zommed

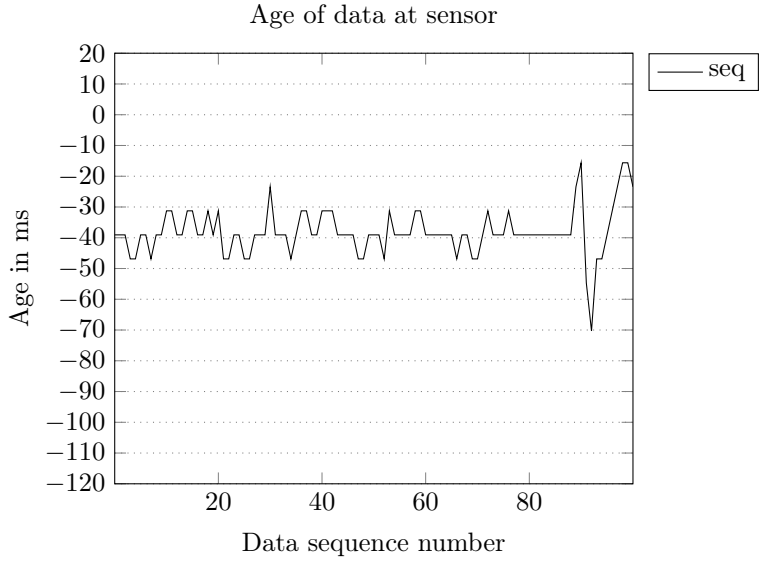

Figure 24: sensor/test/alpha/0.1/rtttarget/2.5/diff/0.98/slope/0.15/zommed

6 Discussion

This section discuss the measurements obtained in Section 4 and 5.

6.1 Performance evaluation

The latency performance evaluation was set up to measure the overhead it takes to compute the CCN communication protocol used as an application for communication between IoT devices. Although the results were impressive overall and gave answers to several questions, they also opened up for further speculations.

Some of the min values that occur are around 40 % less than the median/average value. At first glance, this was thought to be proof that the general latency time could be lowered, but since there are only a few of these outliers there seem to be other reasons why these lower values occur. They appear more often the smaller the data packet is, which indicates that it is linked with something in the transport layer and how the data is packaged.

The round trip time is generally high with ping6 commands at around 25 ms compared with general speeds of 1-2 ms in a local network. One can argue that there are two factors affecting the round trip time. According to Niclas Finne and Joakim Eriksson who developed the Sparrow software, which is the border gateway to the sensor network, it is due to settings in the software that set a delay between each fragment when communicating with the serial-radio. Secondly, it depends on the way pthread is handling some specific polling function. Since both of these factors are caused by the same software in the gateway, regardless of whether CCN or ping6 is used, it is likely that it does not affect the relative conclusions/comparisions made from the latency measurement results.

6.2 Suitability evaluation

The suitablility evaluation was created to show the feasibility of retrieving data from a sensor that is sequentially produced at a fixed interval with the lowest latency possible. This in order to show that the latency properties does not get worse with CCN compared with legacy protocols. The algorithm developed in this thesis was a first attempt to enable such functionality for ICN in IoT networks.

The varying round trips shown in the result section have several explanations. As α grows, the significance of the previous rtt value decreases and instead the importance of the current rtt value grows, which provides less smoothing effect for the srtt. When comparing figure 15, with low α, to figure 17, with high α, one can see that the number of deviations increases as α grows.

A Deviation has to do with when a srtt value was slightly delayed for some reason and becomes compensated at the next iteration in order to 'repar/catch up' that slight delay. One can see this occur at interval 59 in figure 17, where the latency first decreases and in the next interval, it bounce back up and the latency becomes greater. When the delays continues, which it does in the previous example, the algortihm overcompensates which result in a deviation on the other side. On the other hand, when the delay becomes stable after compsensation, the deviation ends, which is seen in figure 15. One of the reasons the deviation are approximatly the same in size, 7-9 ms, is the additional tick it takes to consume a *PIT* entry at the sensor node. The clock resolution on the sensor is 1/128th of a second, which equals 7.8125 ms. This difference is similar to the difference described earlier concerning minimum printing and essential printing in the performance evaluation.

Currently the algorithm has a small startup curve in order to retreive the *rtt_ min*-value before using the PIT of the sensor. After this value is set, there are currently no possibility to recaclulate it or change it later on. This is a known drawback with this algorithm, which could lead to system failure if the *rtt_ min* becomes lower than its initial value. However, problems related to the *rtt_ min*, did not occur during the thesis experiments.

A problem with lowering the *rtt_ min* is when the interval length on the sensor node is a little shorter in comparison to the interval length set in the algorithm. This can occur due to slight inaccuracies in the clock frequency between the sensor and the consumer. As the results show, the system continued to work for differences up to -2%. Initial tests were done with up to -5%. Then however, the system crashed and the algorithm stoped working properly. With a more advanced algortihm where the consumer could adjust its interval based after the interval on the sensor, problems like these could cease to exist.

The algorithm has limited error correcting capabilities and the results do not show any data on how it handles errors. Two issues that can occur are *interest* message timeouts and if the algorithm starts to issue *interest* messages with the wrong sequence number.

It handles timeouts without any greater difficulties for shorter periods of time. Although not shown in the data of the thesis, when a timeout occures, the algorithm continues to successfully send *interest* messages for the next sequence number. This is caused by the serving of the previous correction factor, so it is able to keep the "state" for some period of time. Since the test was not configured to focus on timeouts specifically, no further investigations have been carried out.

There are more problems to recover from when the sequence number gets unsynced between the producer and the consumer. It has not been a task for this thesis to provide any recovery capabilities or the abillity to reconfigurate the minimum latency time. Therefore, it has not been tested specifically either, but initial tests carried out by the thesis, shows that the system tolerates a limited numbers of suchs errors when they occure. Depending on how many data entries the producers *cache* can hold, the consumer can be at most the *cache size minus one* sequence numbers after the current data and still be able to retreive the data. Another possibility is that the size of the producers *PIT* is large enough to handle several incoming requests and that it is able to hold them for until their data is produced.

7 Conclusion

The aim of this study was to evaluate the applicability of ICN in the IoT domain and its performance compared to legacy solutions. It should investigate if CCN is a suitable candidate that potentially could replace for example MQTT as an communication paradigm. The evaluation considered performance measurements of the CCN-lite application components, the overall latency at the system and the feasibility of using ICN in the IoT domain.

The performance evaluation showed that CCN-lite can deliver data from a sensornode to its gateway with low latency. When using the CCN peek application to retrieve data from the sensors CCN-lite implementation, it performs only a few milliseconds slower compared to the ping counterpart which was used as a reference point in the measurements.

The optimization of the CCN-lite code at the sensor shows that part of the long round trip times experienced earlier was due to various debug messages that were printed on the console. Furthermore, the sensor was put into sleep mode for a short period of time for each *interest* message it received. The optimization of the code resulted in a reduction of the round trip times with around 100 ms compared to before. From round trip times of 130 ms to less than 30 ms.

The suitability evaluation showed that ICN is successful in achieving message delivering with a low delay between the creation of a data item and its delivering. When using of CCNs *PIT* and sending the *interest* messages before the data has been created, CCN can eliminate unnecessary waiting time between data creation and sending it. Thereby achiving similar latencies as when using MQTT.

The algortihm presented in this thesis is a first attempt to fetch data periodically produced at a fixed interval using ICN. The algorithm proved to be successful in retreiving data over long periods of time, no timeouts occured during the tests.

As the experiment shows, the algorithm handles high variations in roundtrip times without any issues. It is because of the logics behind the smoothed round trip time that makes the latency values more or less varying. As α grows, the significance of historical rtt-values decreases.

Based on the findings in this thesis, the algorithm handles variances in clock drift to a certain degree. When the data publishing interval at the sensor is greater than the requesting interval of the consumer, the system stays stable. The length of the *interest* message's timeout, at the consumer is the only variable that then can cause the algorithm to become unstable. When the data producing interval at the sensor is negative compared to the consumer the algortihm handles a limited difference in interval length. The difference between the rtt_target and rtt_min has to be be greater than the drift between the consumer interval and the producer, otherwise the system will become instable.

The conclusion drawn in this thesis is that the communication paradigm ICN provides a suitable alternative, in comparision to MQTT, in the IoT domain. A consumer, which was represented as a border gateway, can receive produced data from the producer with the lowest possible latency when it uses the algorithm provided in this thesis.

7.1 Future work

This thesis shows a subset of the performance and suitability of using ICN in IoT networks. There are several other interesting aspects to consider for future studies. In this paper, the communication pattern was between a gateway and a sensor. One of the more critical aspects is how well ICN would perform if the network load would increase and how it would impact the performance overall. In a normal usage pattern, there would likely be several users wanting to access the produced data. There are indications from this thesis that the sensor's processing power is too slow to handle several *interest* messages and deliver responses towards multiple users. It would be interesting to see how the software performs on sensors that have greater processing capabilities than the ones used in this thesis.

The algorithm presented here gives a user the possiblity to retreive data that is sequentially produced at a fixed interval. The interval rate at when the producer creates data is assumed to be known in advance by the consumer. Although this interval does not have to be exact, a rough estimation has to be given, otherwise the consumer's algorithm will not work properly. It would be of interest to se if it is possible to develop an algortihm that can handle larger variations in interval length which is not dependent on parameters such as *rtt_ target*.

One of the main benefits with the presented algortihm resides in the non-existing overhead it takes from data creation until it reaches the consumer, minimizing this time close to zero. However, the round trip time it takes from a consumer's perspective is not optimized today. From its view, an *interest* message potentially spends most time in the PIT during the round trip. Although the *rtt_ target* parameter is specifying how well in advance a user can send the *interest* message, there are no deeper studies regarding how close/low this can be in order to achive a stable system.

Yet another aspect that would be interesting ot investigate is how to handle the 'tune-in' period. Now, client cuts the interval by a small factor each interval until it receives a timeout. Then it 'knows' that it has started to use the *PIT*, thereafter using the algorithm. This can make the starting period take several iterations depending on how long the interval is. There are several different approaches to handle situations like these, but there are no investigations regarding which approach is the best in this case.

From a greater perspective, there are some interesting aspects to consider for future studies. First and foremost, the energy consumption of nodes/the system. This topic has not been covered in this thesis in any way. In order to make ICN a prominent choice for usage in IoT networks, it has to prove to be energy efficient, thereby there has to be further investigations in how well it performs from a energy consumption perspective. MQTT has a general advantage here with its push mechanism, where the sensor can be in sleep mode until it needs to transmit the newly created data. Since ICN does not have the same features, other solutions have to be provided in order to have a small energy footprint. A potential power saving strategy for ICN could be by sending burts of sequences of *interest* messages towards the sensor, stacking them up in the *PIT*. Thereby potentialy becoming more energy efficient.

Another interesting aspect for evaluating is how well ICN performs in comparison to other communication protocols such as CoAP or MQTT within the IoT domian. Not only by evaluating and comparing performance metrics such as protocol overhead times, various of latencies, but also a more deeper investigation regarding how suitable it is to use these communication protocols. Where the other protocols, CoAP and MQTT, provides a long term subscription approach in order to provide consumers with latest data, ICN has to use some kind of short time subscription approach to do the same, potentially giving it more overhead compared to the others. This could test

the limits of the ICN architecture from several perspectives and push the development of ICN as a whole forward.

References

[1] Bengt Ahlgren, Christian Dannewits, Claudio Imbreenda, Dirk Kutscher, and Börje Ohlman. A survey of information-centric networking. *IEEE Communications Magazine*, 50(7), 2012.

[2] Alan Carlton. Information-centric networking could fix these Internet problems kernel description. https://www.computerworld.com/article/3072218/software-defined-networking/information-centric-networking-could-fix-these-internet-problems.html visited 2017-09-20, 2016.

[3] Chip-makers are betting that moores law wont matter in the internet of things. https://qz.com/218514/chip-makers-are-betting-that-moores-law-wont-matter-in-the-internet-of-things/ visited 2017-09-20.

[4] Anders Lindgren, Fehmi Ben Abdesslem, Bengt Ahlgren, Olov Schelen, and Adeel Mohammad Malik. Design choices for the iot in information-centric networks. *IEEE Consumer Communications and Networking Conference*, 2016.

[5] Mqtt. http://www.mqtt.org visited 2017-08-01.

[6] Ccn-lite, content centric netowrking lite platform. http://www.ccn-lite.net/ visited 2017-08-20.

[7] Yanqiu Wu. Adapting information-centric networking to small sensor nodes for hetrogeneous iot network. 2016.

[8] Van Jacobson, Diana K Smetters, James D Thornton, Michael F Plass, Nicholas H Briggs, and Rebecca L Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 1–12. ACM, 2009.

[9] George Xylomenos, Christopher N Ververidis, Vasilios A Siris, Nikos Fotiou, Christos Tsilopoulos, Xenofon Vasilakos, Konstantinos V Katsaros, and George C Polyzos. A survey of information-centric networking research. *IEEE Communications Surveys & Tutorials*, 16(2):1024–1049, 2014.

[10] M. Amadeo, C. Campolo, J. Quevedo, D. Corujo, A. Molinaro, A. Iera, R. L. Aguiar, and A. V. Vasilakos. Information-centric networking for the internet of things: challenges and opportunities. *IEEE Network*, 30(2):92–100, March 2016.

[11] Emmanuel Baccelli, Christian Mehlis, Oliver Hahm, Thomas C. Schmidt, and Matthias Wählisch. Information centric networking in the iot: Experiments with ndn in the wild. In *Proceedings of the 1st ACM Conference on Information-Centric Networking*, ACM-ICN '14, pages 77–86, New York, NY, USA, 2014. ACM.

[12] Riot. https://riot-os.org/ visited 2017-10-10.

[13] Forecast: The internet of things, worldwide, 2013. https://www.gartner.com/newsroom/id/2636073 visited 2017-10-20.

[14] D.R. Cheriton and M. Gritter. Triad: A new next-generation internet architecture, 2000.

[15] Ieee standard for local and metropolitan area networks–part 15.4: Low-rate wireless personal area networks (lr-wpans) amendment 1: Mac sublayer. *IEEE Std 802.15.4-2011 (Amendment to IEEE Std 802.15.4-2011)*, pages 1–225, April 2012.

[16] Ian Poole. Ieee 802.15.4 technology and standard. http://www.radio-electronics.com/info/wireless/ieee-802-15-4/wireless-standard-technology.php visited 2017-07-22.

[17] Internet Protocol, Darpa Internet program protocol specification. RFC 791, September 1981.

[18] Steve Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, December 1998.

[19] Steve Deering and R. Hinden. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944, September 2007.

[20] Urs Hunkeler, Hong Linh Truong, and Andy Stanford-Clark. Mqtt-s—a publish/subscribe protocol for wireless sensor networks. In *Communication systems software and middleware and workshops, 2008. comsware 2008. 3rd international conference on*, pages 791–798. IEEE, 2008.

[21] Andy Stanford-Clark and Hong Linh Truong. Mqtt for sensor networks(mqtt-sn), version 1.2. http://mqtt.org/new/wp-content/uploads/2009/06/MQTT-SN_spec_v1.2.pdf visited 2017-07-28, 2008.

[22] Z. Shelby, K. Hartke, and C. Bormann. The Constrained Application Protocol (CoAP). RFC 7959, June 2014.

[23] Contiki-os. http://www.contiki-os.org visited 2017-07-24.

[24] Sparrow application layer. https://github.com/sics-iot/sparrow visited 2017-08-20.

[25] Texas instruments, cc2650 sensortag. http://www.ti.com/tool/cc2650stk visited 2017-09-20.

[26] Zolertia firefly. https://zolertia.io/product/firefly/ visited 2017-09-20.

[27] Raspberry pi 3. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ visited 2017-09-20.