Evaluating the use of ICN for Internet of things

Johan Carlquist

November 2, 2017

Abstract

Your abstract.

Contents

1 Introduction

The main paradigm of networks today can be called *host centric* and most of our communication is defined as end-to-end between these hosts. It is predicted that in 2020 we reach around 50 billion Internet of Things (IoT) devices [1] and the usage of these devices often imply information centric usage patterns [2].

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 [2] which could lead to improvements in the network as a whole.

Hela introt lite väl kort...

Borde nämna "named hosts" här också.

- Rewrite.
- Write this lastly.

1.1 Motivation

- Remove this and put it under introduction instead?

1.2 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. By evaluating the performance of ICN components and the application as a whole, one can provide a comparision between such an application and a network reference point. Ideally it should be possible to divide and present the throughput, and how much delay time, spent operating under ICN. Therefore a part of the goal of this thesis project is to provide an in depth analysis of where most time is spent when using the ICN application.

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 [3] where the producer sends the newly created data towards a broker, which then pushes the data to a user/consumer. By evaluating the feasibility, a major goal is to investigate if it is possible to achive similar outcome with ICN together with a onetime subscription model and if such a system would be stable. 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.

vad menas med ref point och resterande del av meningen. // A

Formulera om. // A

1.3 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.

1.4 Research methodology

Currently there is a CCN-lite implementation for the IoT light-weighted Contiki OS, the implementation was conducted by a former thesis worker Yanqiu Wu [4]. 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 CC2650, 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 short cycles and small loops between each measurement. This iterative approach has the advantage that it can give answers to questions that are not specifically asked and it can also be used to see changes over time. 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.

Behöver kompletteras med lite mer generell text om experimentell forskning.// B

Berätta att ccn-lite är en lw impl av ccn, sen har yanqiu portat det till contiki.// a

Short cycels of what? // a

1.5 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.

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 that calculates the roundtrip times for IPv6 devices to the CCN-lite project. Moreover, software enabling a consumer to retrieve data from the publisher through a one-time subscription model described 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.

Formulera om, vad är det som menas? // a

1.6 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 [5]. 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[2]. 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[2][6].

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. [7], 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[7].

Behöver koppla named data till IoT-context - Namngivet sensor-data // bengt

Another study by Amadeo et al. [8] 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 [5].

Baccelli et al. were first to port of CCN-lite to the IoT operating system RIOT [9][10]. 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[4]. 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.

Prata mer om hur man hittar senaste värdet. // anders

1.7 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

Describe the greater context, what are the technologies and protocols figuring in this thesis. In beginning mention a little bit of both IoT and ICN sort of waving them together.

2.1 Internet of Things - network stack

- Write about growing number of devices
- how the devices come into play and where they're used.
- Describe the devices similarity, that they often are the same. High level of hetrogenous.

Write about the growing nubmer if devices. How they're coming into play and where they're used.

2.1.1 IEEE 802.15.4

The IEEE 802.15.4 standard intends to offer the fundamental lower network layers for wireless personal area networks (WPAN). The standard focus is on providing a 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 applicant[11] [12]. 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. The communication range is set up to 10 meter and a 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[12]. Full functional device(FFD) and reduced function device(RFD). A FFD node implements all communication funcionallity 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 cooardinator. 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 functionallity 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 topogoly, 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 and the distance a node can cover is set to be at a maximum of ten meters.
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

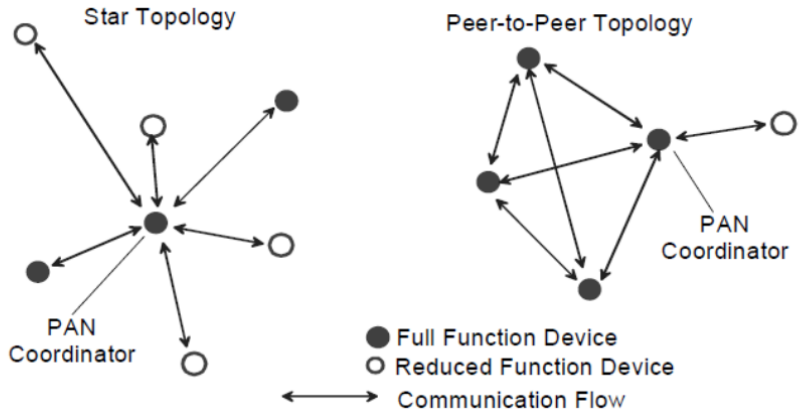

Figure 1: Topology structures in 802.15.4 networks. Star structure on the left, Peer-to-peer on the right.

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.

2.1.2 IPv4, IPv6

Since the introduction of Internet Protocol version 4 (IPv4) in 1981[13], 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[14]. The IPv6 protocol defines the length of an address of 128 bits, which lead to a total 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 need can not be met, fragmentation and reassembly must be provided at layer below IPv6[14].

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) [15]. Futhermore, the IPv6 header is 40 octets long which is almost a third of the total 802.15.4 MTU (at least 25 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 figure 2.

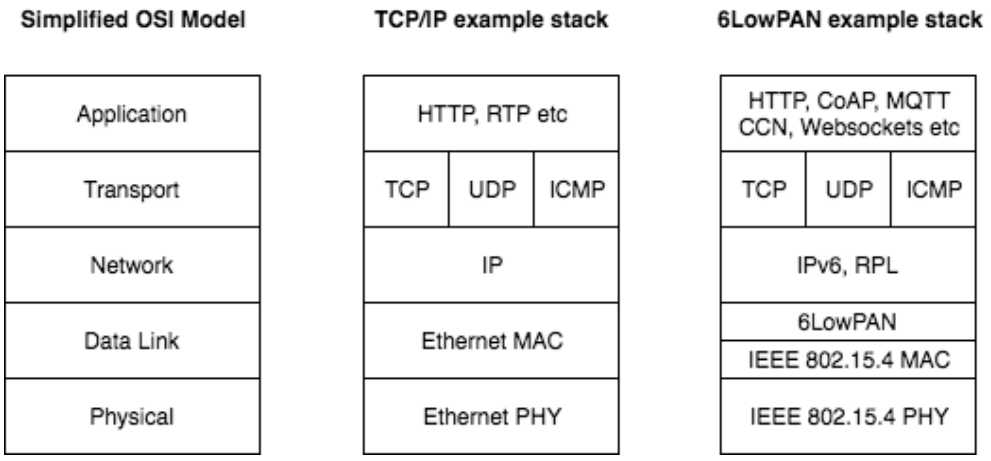

Figure 2: The simplified OSI model, an example TCP/IP stack and an illustration of an example 6LowPAN protocol stacks.

To address these issues, among several, "IPv6 over Low-Power Wireless Personal Area Networks" (6LoWPAN) was established in 2007 as an adoption 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) [15]. 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 performance reasons it is 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 unreliable networks targeting Machine-to-Machine (M2M) communication[3]. The protocol reside in the application layer in the OSI model, assuming that the underlaying network structure provides a point-to-point, session-oriented data transport provided by example TCP/IP [16]. 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 interested 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 publisher. The broker is usually located in a traditional network due to its higher performance regarding bandwith and processing capabilities.

MQTT-SN, where the extension stands for sensor network, is a MQTT version that is adapted for wireless communication. It is optimized to be implemented on low-cost, battery-operated devices with limited or constrained storage and processing capabilities[17], perticular 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.

[more about brokers and its role.]

2.1.5 CoAP

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol to be used with constrained devices and networks, and between M2M communication[18]. 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 transportation 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 functionallity 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 bit out of 32 bit 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. In this section will continue describing the gerneral ideas. However, the thesis overall will only consider the *Content Centric networking* approach described more in depth at section [ref till CCN].

Some of the main features in the ICN architecture are the possibility of in-network storage for caching data, multiparty communication through replication, decoupling senders and receivers, and that data is named[2]. In ICN networks, an information request may not only be satisfyed by locating the original information source, it is

possible for any of the in-network caches to reply 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 is driven by consumers of data. There are two types of packets 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* packet is only sent in response to an *Interest* message, upon response, the interest message will cease to exist in the network.

A key feature of CCN is that the content names are hierarchical. This allows name resolution and routing information to be aggregate which in turn is critical in order to scale the network. The content name can be similar to the way we access 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 hit 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. Another advantage with periodically and sequencing, is that it provide the possibility to ask for data that has not yet been produced. A consumer could potentially issue an *interest* request a short time before the *data* has been produced, which would firstly get the data directly to the user and secondly minimise the latency on the network[ref to bengt2](No performance evaluation of this has been done to date, the thesis will try to answer the feasability of this.).

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* request recieved by the CR, their current state and a mapping to which face they came from[Bengt]. When the data packet for a particular *interest* arrives to a CR, the data packet will be forwarded back on a reverse path, towards the face that exist in the corresponding PIT entry. After the data packet 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 request[?].

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 an limited period of time for futher use. Although there is 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 object that is requested at most once, since the benefits only occur when the data is requested a second time [7][2].

An example of CCN in action is illustraded in figure 3. Here, a subscriber wants to retrive the indoor temperature data from the producer. Subscriber1 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 subscriber1.
When subscriber2 later on wants the same content, he 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 are the data and services and it is less important to communicate with a specific device [7]. 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 battery and network bandwidth of an IoT device.

2.2.3 CCN-lite

CCN-lite is a lightweighted, functionally interoperable implementation of CCN [19]. 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, without any compromise of the CCN functionallity. CCN-lite runs over UDP and Ethernet, and support packet fragmentation.

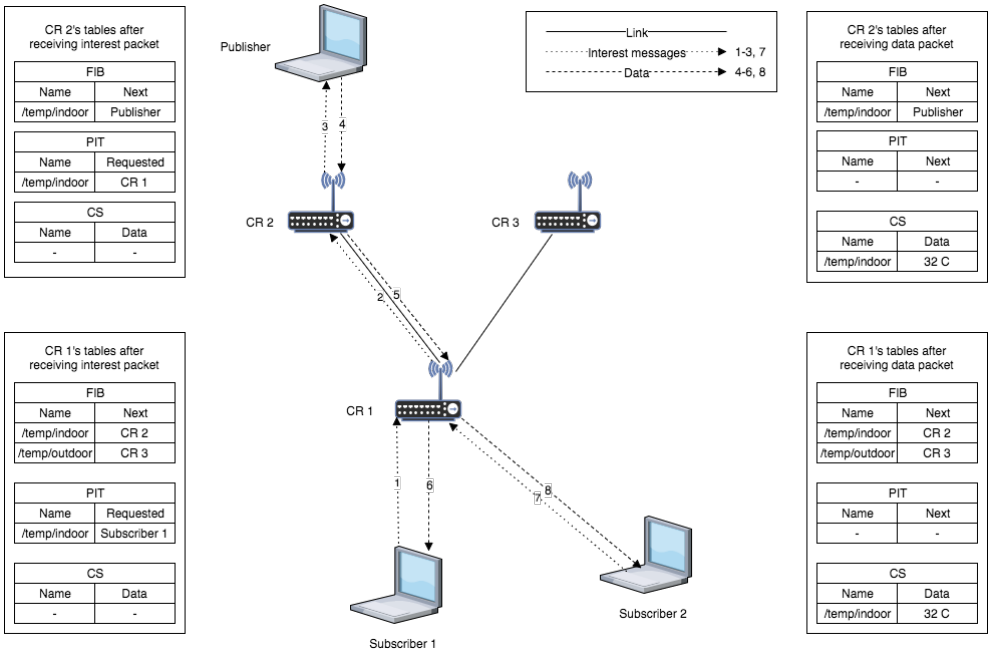

Figure 3: The CCN architecture builed up by content routers (CR), forwarding information base (FIB), pending interest table (PIT) and content store (CS), inspired by [ref survey ICN]

2.3 Contiki-OS

Contiki OS is an open source operating system that is suitable for network-connected, memory-constrained devices that focusing on Internet of Things [20]. It focus on wireless technologies and implement 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 extrem low-power systems.

2.3.1 Sparrow

Sparrow is a commnunication format that encapsulates different types of payload on top of IPv6/UDP [21]. 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 [19] to the Contiki-OS platform in 2016 [4]. The portation enables a sensor to send and receive its application data with the ICN communication pattern providing all the functionallity that the regular CCN-lite provides. A requestor can send an *interest*-request 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

- Utveckla problem statement
- This is the problem
- Continue more in depth than problem statement in chap 1.
- That leds us to these experiments and why theyre done.

3.1 Setup

Some info about why this is the setup. Maybe because of the previous section.

- Ingress.
- Create a picutre that connects the dots, refere to this picture in the subsections.

3.1.1 Hardware

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

3.1.1.1 Texas Instrument CC2650 SensorTag

The CC2650 sensortag is a wireless microcontroller developed by Texas Instruments [22]. 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 [23]. 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 [24]. 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, Sparrow 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* requests and recieve the *data* for the request it has sent. To issue an *interest* request, the CCN 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 recieves 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 times for creating, sending an *interest* and recieving the *data*. 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 responded.

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 to produce 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 the data will be responded toward the issuer. Otherwise, the *interest* will become an entry in the *pending interest table* and no respond toward the user until the requested data has been produced.

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 PIT 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 send 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 comes together and cooperate.

When a CCN peek request is issued, it goes from 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 or discarded. When the sensor responds with data, it will go on the same route backwards to the process on the router that made the CCN peek request initially.

All ping, CCN peek and data messages is sent through the 802.15.4 radio network. The sensor connects to the border router with 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 of 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 there transportation 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 produce and make them readable for a user.

After a few experiments were made, the result was that the amount of information printed on the console had a huge impact on the latency. After iterating a couple of times, 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 or not. For the essential printing, information regarding *interest* arrival times, *data* departure times and other metrics values essential to the result were added. The consequence is that this cost a little on the performance on the sensor, one clock tick on the sensor (1/128th second) more than the minimum printing.

4 Latency performance evaluation

The purpose of this experiment is to measure the roundtrip time, latency, between a sensornode and a border gateway using ping and CCN peek commands. The results show which of these 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. 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 to 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

The roundtrip time is be measured using the ping6 command live tool which uses the ICMPv6 protocol. A similar tool has been developed to measure the latency for a CCN-peek request. Both tools are used in the similar way as in figure 4, 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 seen in figure 4, one can translate the roundtrip-calculation into the equation:

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

$$\Leftrightarrow$$

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

$$\Leftrightarrow$$

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

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.

Ping and peek differs how the total packet size transmitted over the radio is choosen. With the Ping approach, one adds an extra data payload 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 naming of the data to a suitable length. 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

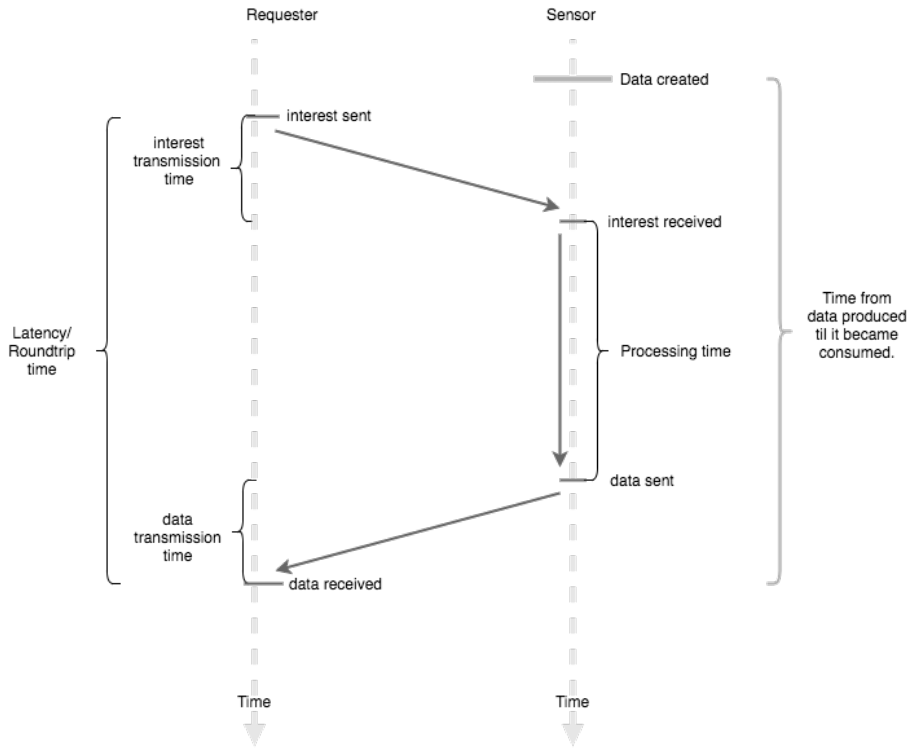

Figure 4: The roundtrip measurement is the time it takes for a client to initiate a request, and get data as a response. Transmission time is the time the bits spent on the wire or radio. Processing time is calculated from the time an interest was received til it was sent back to the requestor.

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 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 times is therefore the same in equation 1 and 2.

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 their total size will be on the 802.15.4 radio link. The total packet range 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.

4.2 Results

The results from the latency times with ping6 are shown in figure 5, 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 6 and 7, 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 5, 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 6 is little above 25 ms for packet sizes the fragmentation limit. Roundtrip times when debugging is turned on, illustrated in figure 7, show a median of around 50 ms. The 20-25 ms difference (around 100%) 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 massive amount of information toward 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 5 and figure 6 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

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

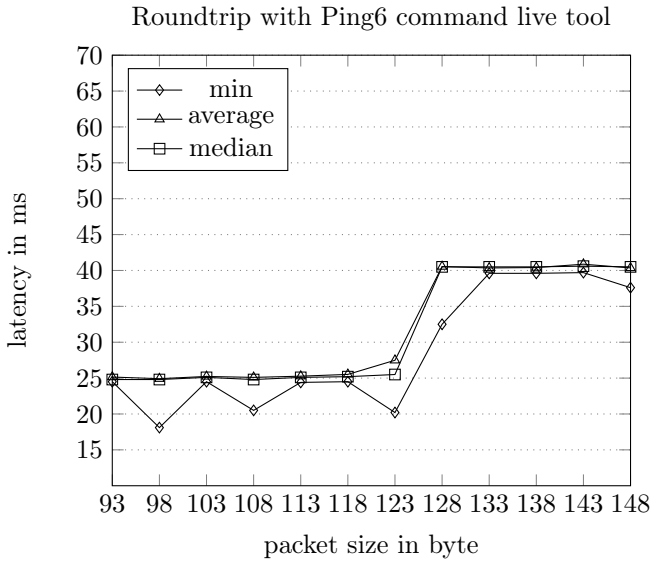

Figure 5: Latencies in milliseconds when pinging a sensornode with packet sizes from 93 byte to 148 byte transmitted over the radio. The latencies stays stable at around 27 ms up until 123 byte. After fragmentataion, the latency raise and stay stable at around 43 ms.

latency and the average/median latency. The histogram for the ping and peek latency measurements are illustrated in figure 8 and 9. 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.

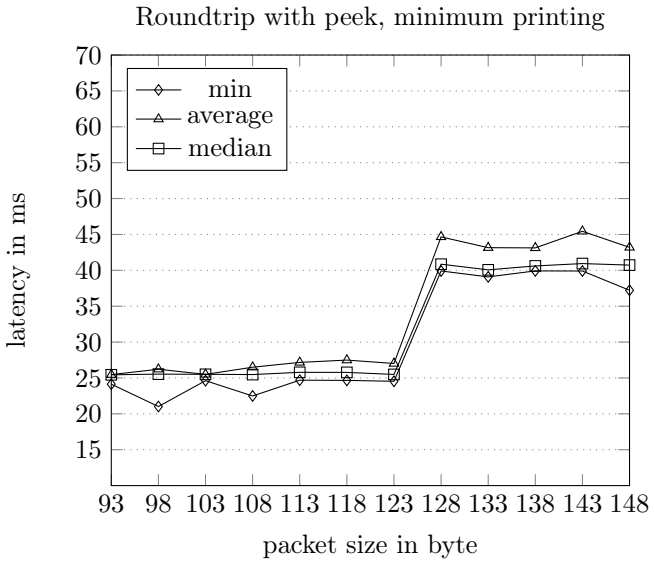

Figure 6: Latencies in milliseconds for peek interest requests of sizes from 93 byte to 148 byte. The latencies stays stable at around 26 ms between 93 byte to 123 byte. After fragmentation, the latency raise and stays stable at around 41 ms.

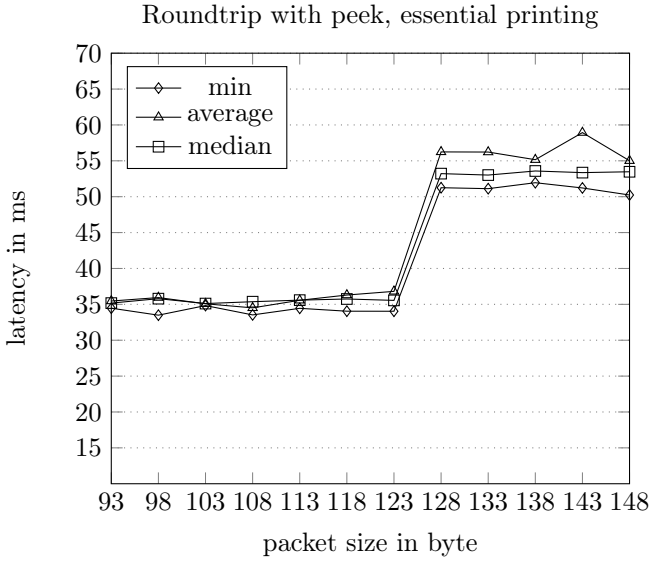

Figure 7: Latencies in milliseconds for peek interest requests of sizes from 93 byte to 148 byte. The latencies is stable at around 35 ms between 93 byte to 123 byte. After fragmentation, the latency raise and stays stable at around 53 ms.

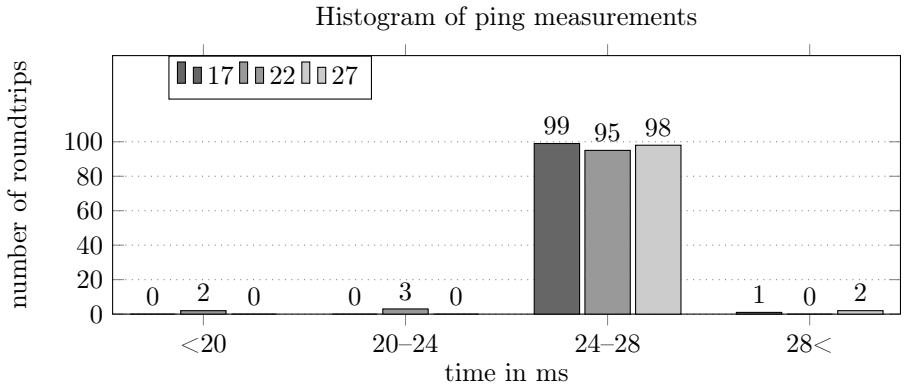

Figure 8: Histogram of ping latencies result. A majority of the roundtrips end up in the 24-28 ms span.

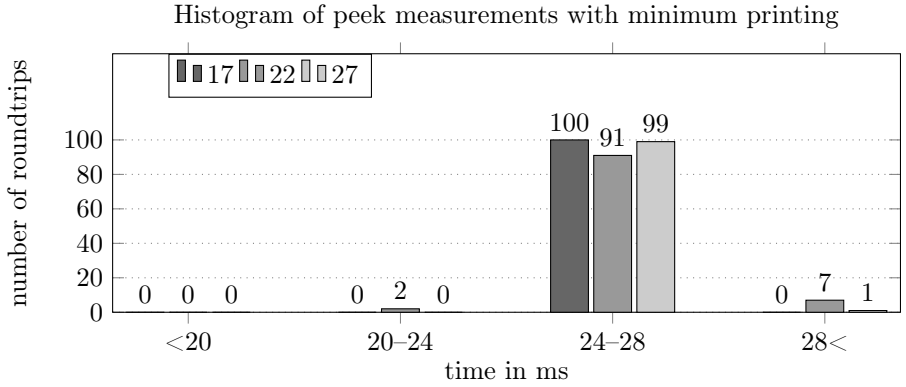

Figure 9: Histogram of CCN-peek latencies result. Almost all of the roundtrips ended up in the 24-28 ms interval.

5 Suitability evaluation

The purpose of this experiment is to create a suitable software for a consumer toat follows an interval and retrieves the data which is created periodically by a sensor node. A key goal is enabling the consumer to recieve the data as soon as it has been created at the sensor. The interval when the data is created could be of arbitray, but fixed, length. Usually, a publish subscribe communication such as MQTT handles this by pushing the data towards the consumer as soon as it is created. With the CCN possibility of storing *interest* requests at the sensors combined with the 'one-time' subscription approach, described by Ahlgrens et al in [7] where the consumer sends an *interest* in advance for future data, it could be possible to achive the same functionallity with ICN. The results show how such an system would perform over time and if it is a feasible replacement to the publish subscribe approach.

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.

5.1 Method

The algorithm developed in this project, shown in figure 10, is a first attempt to retreive data with the previously described one time subscription approach. The algorithm tries to fetch data at a certain interval in order to recieve the data as soon as it has been created on the sensor. 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 makes the system more tolerant when handeling a latency value differs substantially from usual. One must set a round trip target (rtt_target) that the algortihm follows. 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 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.

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

```
next = reference time;
rtt_min;
rtt_target = rtt_min * x;
α = 0 <= α <= 1;
while infinity do
    rtt = send_interest_receive_data;
    if not timeout then
        srtt = α × rtt + (α - 1) × srtt;
        corr = srtt - rtt_target;
    end
    next_time = next_time + interval_time + corr;
    sleep(next_time - current_time);
end
```

Figure 10: Algorithm that makes it possible for a consumer to follow the creation of data with a certain interval.

early towards the sensor. At the same time, if the srtt latency becomes to short, the request is sent to late.

5.2 Result

The results regarding latencies when using $\alpha = 0.1$ and $rtt\_target = 2.5$ is shown in figure 11. 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/consumers perspective. The results, illustrated in figure 11, shows that the latency is stable between 90-100 ms, the srtt is stable with a small variance around 95 ms and that the correlation 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 12, 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 12, 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.

When α is changed to 0.9, the resulting curves, illustrated in figure 13 and 14, show more alternating forms. The latencies range from 87 ms to 117 ms, and most ofthen they are either at the both ends spectrum or at 97 ms. The age of the *interests* is at the same time pending 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 rtts start to alternate, it is hard for the system to stop.

When α is set to 1, illustrated in figure 15 and 16, the latency times alternate between 87 ms and 117 ms as seen with $\alpha = 0.9$, but here the frequency is greater. In this case, the smoothing is not available and therefore the correction is only dependent on the last roundtrip time. 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.

5.2.1 Different intervals at the sensor

All of the previous results were made under the assumption that the two intervals are relatively the same. One interval second on the sensor is almost the same as one interval second on the GW. To illustrate the functionallity when the sensor is providing data at different periodicity than the GW, other tests have been made. The algorithm proves to stay stable even for variances in the drift time caused by the sensor.

In figure 17 and 18, 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 is a positive drift in time on the sensor. The time it takes for an *interest* to be consumed by the sensor is around 78 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 logic 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 19, and the time it took to consume the interest was around 40 ms as seen in figure 20. 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.

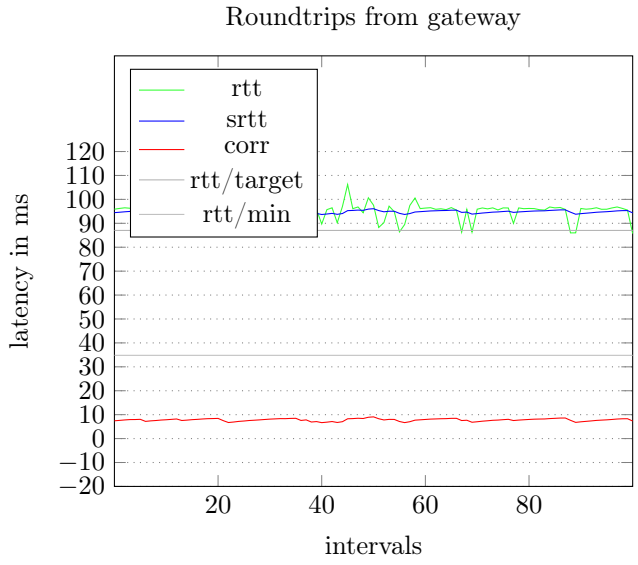

Figure 11: gw/test/alpha/0.1/rtttarget/2.5/diff/1.0/slope/0.15/zommed

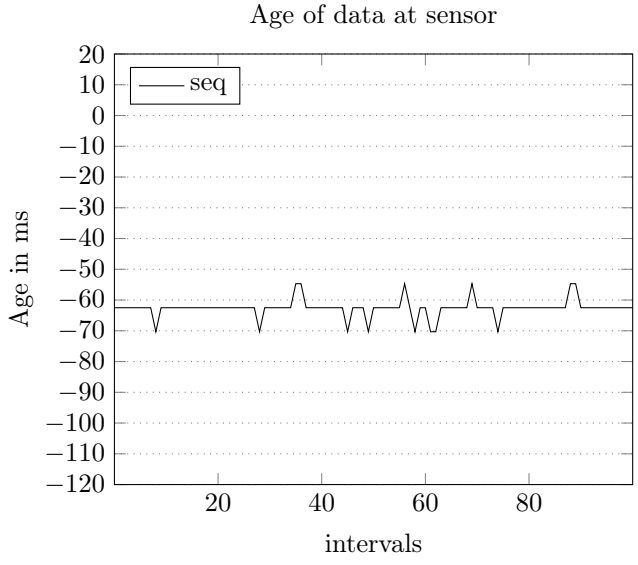

Figure 12: sensor/test/alpha/0.1/rtttarget/2.5/diff/1.0/slope/0.15/zommed

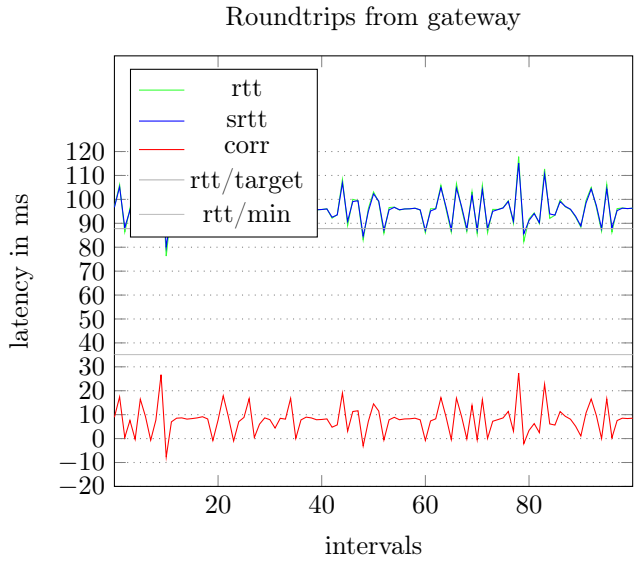

Figure 13: gw/test/alpha/0.9/rtttarget/2.5/diff/1.0/slope/0.15/zommed

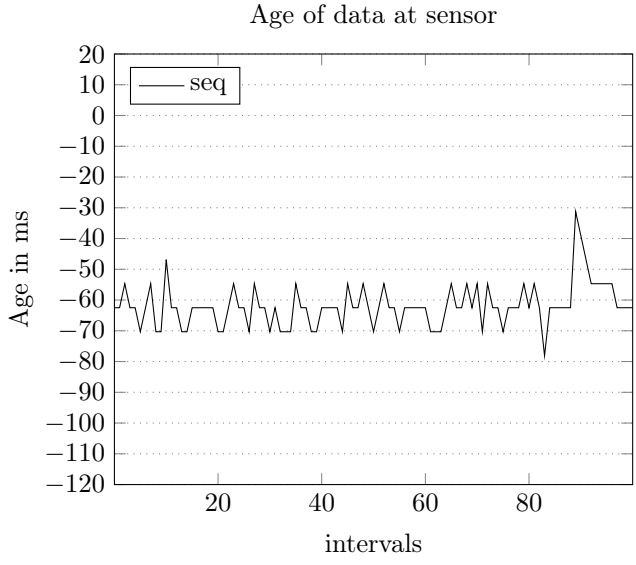

Figure 14: sensor/test/alpha/0.9/rtttarget/2.5/diff/1.0/slope/0.15/zommed

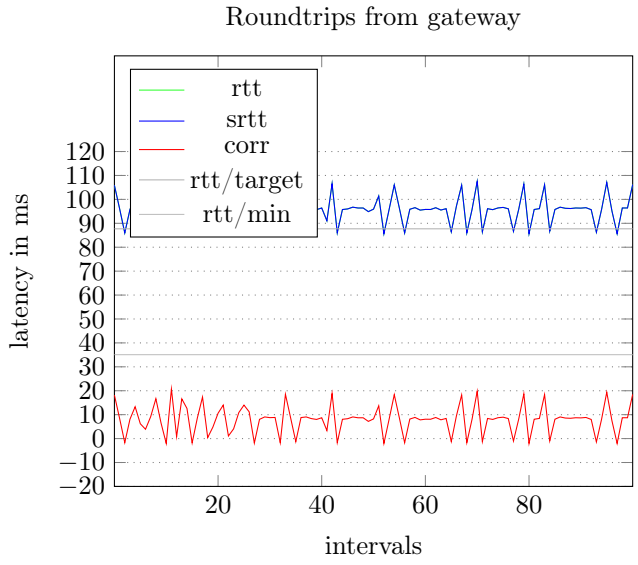

Figure 15: gw/test/alpha/1.0/rtttarget/2.5/diff/1.0/slope/0.15/zommed

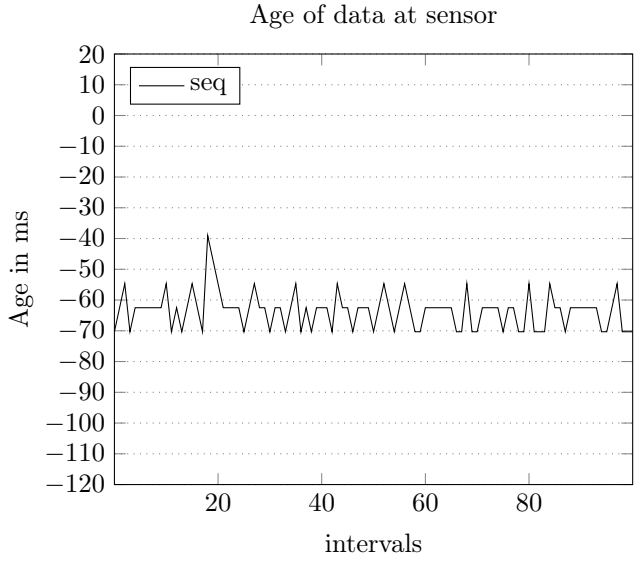

Figure 16: sensor/test/alpha/1.0/rtttarget/2.5/diff/1.0/slope/0.15/zommed

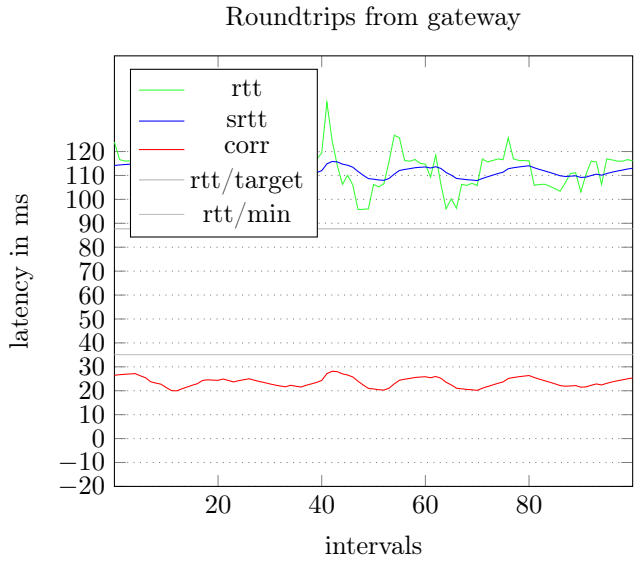

Figure 17: gw/test/alpha/0.1/rtttarget/2.5/diff/1.02/slope/0.15/zommed

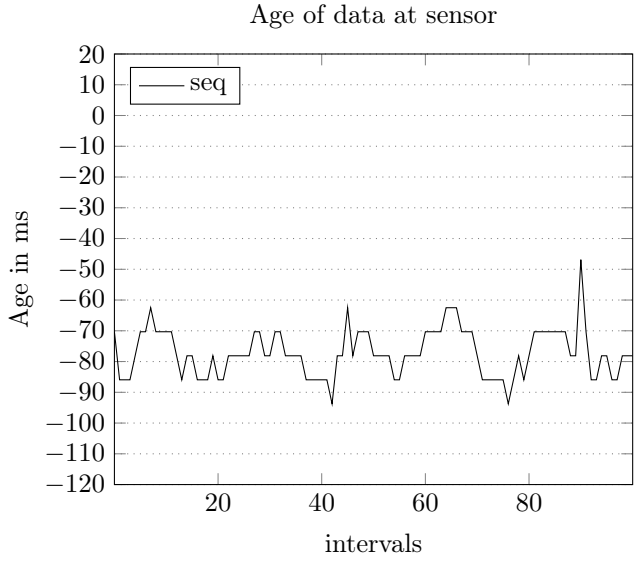

Figure 18: sensor/test/alpha/0.1/rtttarget/2.5/diff/1.02/slope/0.15/zommed

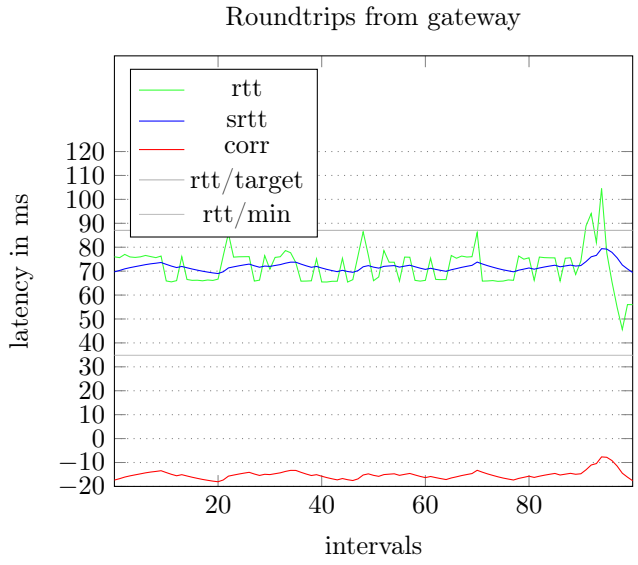

Figure 19: gw/test/alpha/0.1/rtttarget/2.5/diff/0.98/slope/0.15/zommed

Figure 20: sensor/test/alpha/0.1/rtttarget/2.5/diff/0.98/slope/0.15/zommed

6 Discussion

- Questionize your result
- Arugment that it is cool.

7 Conclusion

- Summarize your contributions
- Conclusions from the results
- Implications for the future
- Be breif!
- Its a suitable system. Works flawlessly.

7.1 Future work

References

[1] Alan Carlton. Information-centric networking could fix these Internet problems kernel description, 2016.

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

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

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

[12] 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.

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

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

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

[16] 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.

[17] 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.

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

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

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

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

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

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

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