Content-Centric Networking in the Internet of Things

Otto Waltari

Helsinki 25.11.2013

M.Sc. thesis

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Otto Waltari

Työn nimi — Arbetets titel — Title

Content-Centric Networking in the Internet of Things

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
M.Sc. thesis	25.11.2013	45 sivua

Tiivistelmä — Referat — Abstract

Advanced low-cost wireless technologies have enabled a huge variety of real life applications in the past years. Wireless sensor technologies have emerged in almost every application field imaginable. Smartphones equipped with Internet connectivity and home electronics with networking capability have made their way to everyday life. The *Internet of Things* (IoT) is a novel paradigm that has risen to frame the idea of a large scale sensing ecosystem, in which all possible devices could contribute. The definition of a *thing* in this context is very vague. It can be anything from passive RFID tags on retail packaging to intelligent transducers observing the surrounding world. The amount of connected devices in such a worldwide sensing network would be enormous. This is ultimately challenging for the current Internet architecture which is several decades old and is based on host-to-host connectivity.

The current Internet addresses content by location. It is based on point-to-point connections, which eventually means that every connected device has to be uniquely addressable through a hostname or an IP address. This paradigm was originally designed for sharing resources rather than data. Today the majority of Internet usage consists of sharing data, which is not what it was originally designed for. Various patchy improvements have come and gone, but a thorough architectural redesign is required sooner or later. *Information-Centric Networking* (ICN) is a new networking paradigm that addresses content by name instead of location. Its goal is to replace the current *where* with *what*, since the location of most content on the Internet is irrelevant to the end user. Several ICN architecture proposals have emerged from the research community, out of which *Content-Centric Networking* (CCN) is the most significant one in the context of this thesis.

We have come up with the idea of combining CCN with the concept of IoT. In this thesis we look at different ways on how to make use of the hierarchical CCN content naming, in-network caching and other information-centric networking characteristics in a sensor environment. As a proof of concept we implemented a presentation bridge for a home automation system that provides services to the network through CCN.

ACM Computing Classification System (CCS): C.2.2 [Network Protocols]

Avainsanat — Nyckelord — Keywords
Content-Centric Networking, Information-Centric Networking, Internet of Things

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Contents

Abbreviations and acronyms

API	Application Programming Interface
CCN	Content-Centric Networking
CO	ContentObject
CoAP	Constrained Application Protocol
CS	Content Store
EMF	Electromagnetic Field
FIB	Forwarding Information Base
HTTP	Hypertext Transfer Protocol
ICN	Information-Centric Networking
IM	InterestMessage
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
M2M	Machine-to-Machine
MqTT	MQ Telemetry Transport
PIT	Pending Interest Table
REST	Representational State Transfer
RFID	Radio-frequency Identification
SW	StartWrite
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WSN	Wireless Sensor Network

1 Introduction

Advanced low-cost wireless technologies have enabled a huge variety of real life applications in the past years. Wireless technologies have emerged in almost every application field imaginable. Any subject that requires surveillance, monitoring, telemetry, or telecommand is a plausible target for wireless sensor applications. Low-cost technologies has made these sensor networks affordable, and thus available even for consumers. Home automation, theft alarm, surveillance, monitoring, and other smart applications at a reasonable cost is something that attracts consumers.

Many of the sensor networks today are completely segregated and isolated. In critical applications that is intentional and desired in terms of security and safety. While on the other hand, many sensor networks could contribute, or benefit from being connected to a bigger ecosystem. Extensive connectivity of the sensor network nodes is required in order to participate in a large ecosystem, which brings us to the essence of this thesis.

A novel paradigm has risen to frame the idea of a large scale sensor ecosystem. It is called the *Internet of Things* (IoT) [AIM10], in which the things stand for anything that is connected to the Internet. Connecting a huge amount of various devices to the Internet, however, challenges the current state of the worldwide network. We see that there are two main challenges; connectivity and communication.

Connectivity backbone in today's Internet is the *Internet Protocol* (IP). The IP paradigm dates back several decades and it was originally designed for sharing resources rather than data. It is based on point-to-point connections, which eventually means that every connected device has to be uniquely addressable through a hostname or an IP address. Connecting billions of devices this way requires an equal amount of allocated IP addresses. The dominating IP address space, IPv4, was depleted in 2011 [ICA11]. Its follower, IPv6, is making its breakthrough at a relatively slow pace due to various technologies [DDWL11] that aim to extend the lifetime of IPv4. While the current state of the IP architecture complies with most of today's Internet usage needs, there are use cases in which a different approach could work better.

Communication is another challenge we have to consider while dealing with a huge amount of devices. It is important that the amount of data several billion devices can produce is easily achievable in the network under any circumstances. Most current communication protocols in IoT rely on point-to-point connections and are vulnerable to link breakdowns. Many of them also use data storages and broker servers, which introduce potential single point of failures, unless replicated sufficiently. Also, we must not forget that hardly any of the current protocols are compatible with each other. Protocol incompatibility drives the IoT concept towards a sparse bunch of separate sensor networks.

We have come up with bringing *Information Centric Networking* [XVS+13] (ICN) to the Internet of Things concept. ICN is a new networking paradigm which tries to move device connectivity away from the point-to-point model familiar from IP. It is currently in general a hot topic, and several implementations of ICN proposals have emerged. One of them is *Content-Centric Networking* [JST+09] (CCN), which we will look at in more detail later in this thesis. CCN in particular supports in-network storage and transparent in-network caching, which we will prove both to be useful in an IoT environment.

1.1 Problem definition

As mentioned earlier in the previous Chapter, we see that there are two main challenges regarding the current state of the Internet and the future vision of the IoT concept. These two challenges are connectivity and communication. In this Chapter we formulate the problem we see in the current situation. In order to make the problem statement as clear as possible we try to be very concise.

I. Connectivity

The dominating IP paradigm is all about point-to-point connections. We don't see this as a feasible connectivity model for IoT because of the following reasons:

- Each device has to be universally addressable from a limited address pool.
- Point-to-point connections rely heavily on OSI-model data link layer.

II. Communication

Communication protocols rely heavily on the connectivity model below them, which means that some of the problems listed here are reflected from the connectivity problems mentioned above. However, we see the following problems with current communication protocols used in the IoT:

- Several similar competing protocols.
- Gateways and proxies requires for seamless interoperability between competing protocols.
- Centralized data storages.
- No transparent in-network caching.

Some of these problems have already existing solutions. However, we don't see that the solutions would have been successful, or actually solved the problem. We will briefly survey existing solutions in Chapter 4. Afterwards we will propose an alternative solution our way.

1.2 Research goals

At the highest level, our goal is to find solutions to the things listed in Section 1.1. We try to achieve this through implementing a functional communication protocol for sensor networks on top of CCN. Our focus is more on the practical functionality rather than theoretical limits and boundaries. Theory of course is taken into consideration with all respect, but at this point we are more into showing how CCN would work in IoT in practice. In Chapter 5 we present the implementation that we created to achieve our goals.

To summarize, our goal is to show that a communication protocol for IoT is possible to implement with the following features:

1. No point-to-point connections.
2. Transparent in-network caching.
3. In-network storage of sensor data.

4. Reduced workload for the sensor devices.

5. High-level abstraction layer to access sensor devices.

1.3 Thesis outline

This thesis is structured in a logical way to provide the reader all the required preliminary knowledge before going into technical discussion and implementation details. In this Chapter we have introduced the topic of this thesis, pointed out some issues with the current state of art and given some ideals or partial solutions to the issues. In Chapter 2 we give an overview of the IoT. Chapter 3 gives an introduction to the ICN paradigm. We take also a closer look at CCN and its open source platform CCNx, which is a promising implementation of the ICN paradigm. In Chapter 4 we discuss how to benefit from the CCN paradigm in the field of IoT. Chapter 5 explains the testbed implementation we did in order to see CCN in action in a sensor environment. At the end of Chapter 5 we explain the experiment methodology and evaluate its outcome. Finally, in Chapter 6 we conclude the topic and give some final thoughts.

2 Internet of Things

The *Internet of Things* (IoT) [AIM10] is a novel concept of a large-scale wireless sensing ecosystem. The definition of a *thing* in IoT is ambiguous. In a nutshell it stands for something that produces or contributes information with some value to the ecosystem. Practically speaking a *thing* can be anything that is equipped with appropriate technology to make it part of the smart network. These *things* provide endless opportunities for applications in various fields, such as smart cities, -homes, industry, health and transportation. Most of the required technology already exists and harnessing of this great potential is in progress.

The original idea [Ash09] behind IoT emerged from the thought that most of the information moving on the Internet is produced by human beings. People in general observe real life things and generate content based on the observations. However,

human beings have limited time, accuracy and attention, and thus they are not very good at feeding information to the Internet. This statement is quite harsh if we consider content like photographs, music, video, and other kinds of information that requires creativity characteristic to human beings. On the other hand, several other kinds of information related to real life things need no creativity. Information that is easy to achieve, present, and reason by today's technology is much more effectively produced by sensors of different kinds.

In [AIM10], Atzori et al. divide the entire IoT paradigm into three different visions; things-, Internet- and semantic-oriented visions. These visions are driven by different communities that focus on opportunities they have interest in the entire IoT field. Since the IoT as a concept is very wide, this division is very useful in helping readers understand the overview through smaller portions. Figure 1 illustrates this three-way division.

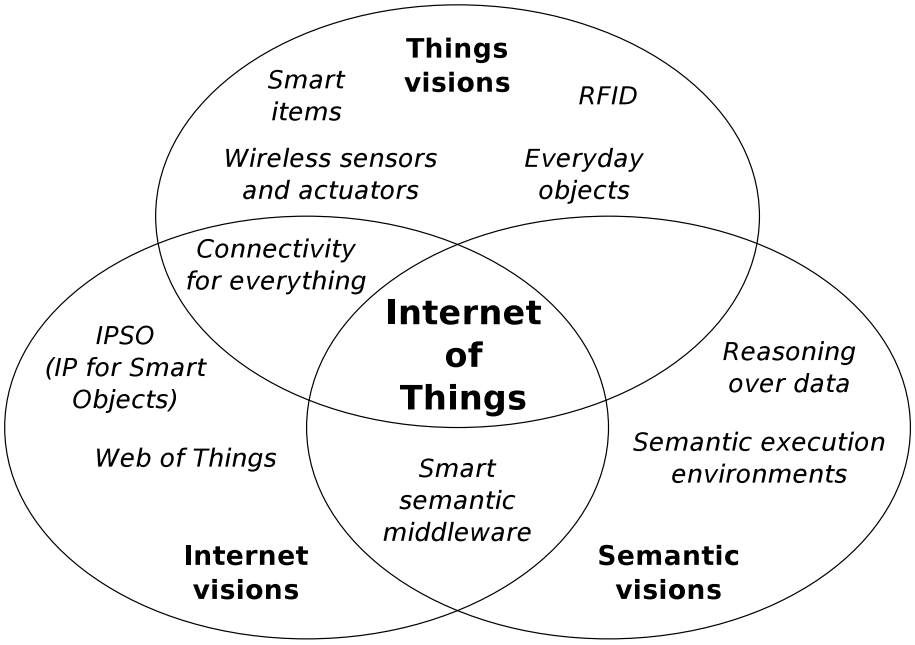

Figure 1: A simplified diagram showing the division of IoT by Atzori et al. [AIM10].

Instead of discussing overlapping visions in a Venn diagram, we divide and represent IoT as a stack of layers. Since the intersection of 'Things visions' and 'Semantic visions' is empty, there is no need to visualize IoT the way it is visualized in Figure 1. The intersection will also always stay empty because a connectivity layer is required

between the things and the semantics. In other words, they are not directly connected by anything, nor will they be. Therefore we feel that a stack representation of IoT is more constructive and easier to approach. We also split the 'Internet visions' into two sections, since we see that it consists of two clearly distinct parts. Our stack division is illustrated in Figure 2. Its terminology and idea is based on the work by Atzori et al. In the rest of this Chapter we will approach each of the layers individually starting from the bottom layer.

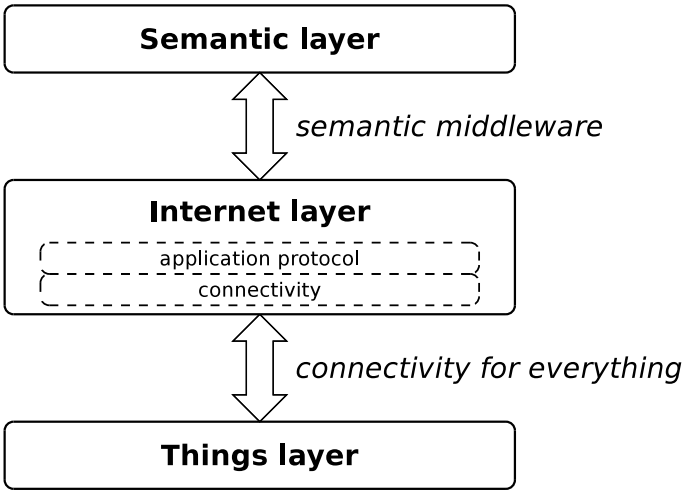

Figure 2: Internet of Things paradigm illustrated as a stack.

2.1 Things layer

In IoT things are the content producers. They can be seen as the leaf nodes gathering data from all edges of the network. Alternatively they can be considered as the interface between the real world and network. As the definition of a thing is ambiguous we provide some examples and use cases for things in this Chapter.

Things in IoT can be either active or passive. A good example of passive things are items equipped with *Radio-frequency identification* (RFID) tags. RFID tags are cheap and the technology is mature and well established. Passive RFID tags do not need any power source. They operate with the power supplied by the readers electromagnetic field (EMF). The tag starts to emit its data when it is exposed to a EMF. Data is written onto the tag in the same manner but with a specific signaling frequency. A common passive RFID tag can store data up to 96 bits and

their range is limited to a couple of meters due to the limited powering harvesting through EMF.

Food supply chains have adopted the usage of passive RFID tags [KRS+09]. Effective handling and inventory management of products is crucial when it comes to spoiling goods. Food crates and pallets have been equipped with RFID tags to reduce human interaction and make the supply chain more efficient. Tags attached to food pallets can also log the environment in simple ways, such as long periods in warm conditions or freezing. Nanotechnology has even made it possible to produce small RFID tags that can be attached to the food itself [TBY+12]. These edible tags change their output based on surface changes of the product it is attached to. Optimizing the supply chain is a way to cut down food waste [MM05], which is a continuously growing global problem.

Another field where RFID tags have been widely adopted is retail market. RFID tags on product packaging may some day even replace the conventional IAN barcodes on items [Wyl06]. Remotely identifiable tags on products allow inventory monitoring in real time, which combined with a smart inventory system could save companies from out-of-stock situations and help in keeping the inventory size as small as possible [MM05]. Tags in retail product packaging allows also automatic payment through smart shopping carts [Rou06]. Such carts may come with an onboard computer that monitors what the customer has in cart and reports it at checkout to the cashier, which might instead of a human be just an automated paying machine.

As RFID tags are inexpensive they can be applied to almost anything; vehicles, commercial goods, food packaging or even animals for identification purposes. All of these real life things can be considered as things in the IoT in case their RFID data can be automatically read and submitted to serve a larger ecosystem.

Active *things* are another category of leaf nodes in the IoT. These can be generalized as sensor devices that observe the surrounding world. As we can not put RFID tags on the weather and read its state for example, we need various kinds of sensors to measure parameters such and temperature, humidity, wind, and so on. Wireless sensors in most cases rely on the same RFID technology, but in contrast to passive tags, active devices are power source equipped and they are capable of transmitting

data on their own through the RFID tag.

In the simplest scenario an active RFID node transmits its data, and within range there is a coupled counterpart that picks up the signal. The maximum distance for this kind of device-to-device communication varies widely because of utilized technology, application specific antennas, transmission power, location and other environmental conditions. The maximum device-to-device distance in most wireless RF-based sensor technologies is usually between 10 to 100 meters. Longer distances can be achieved through several advanced *Wireless Sensor Network* (WSN) technologies that support different network topologies and are capable of multihop routing within the WSN. Many of the affordable and RF-based WSN technologies have similar characteristics since they comply to a common standard such as IEEE 802.15.4 [CGH+02].

Private WSNs is the most common way to add *things* to the Internet. ZigBee [Far08] is probably the most widely adopted commercially available WSN technology. Its cost and power efficiency makes it popular in home automation and other private sensors environments. It is also based on the IEEE 802.15.4 standard. ZigBee was created through the collaboration of a HomeRF spin-off company and IEEE 802 workgroup. Its focus in design is primarily on simplicity, low cost and power efficiency.

In most applications ZigBee operates on a sub-gigahertz frequency. Depending on the continent regulations, it operates either on 868 or 915 MHz band. Its specification allows also operation on the already quite crowded 2.4 GHz frequency. That is, however, quite frivolous in many cases because of a higher power consumption and an unnecessarily high data rate of 250 kb/s. With the commonly used sub-gigahertz frequencies ZigBee is capable of reaching data rates from 20 to 40 kb/s.

ZigBee network topology is a mesh with one central coordinator node. Due to the nature of a mesh network where intermediate nodes can pass messages further, the coverage of a ZigBee network can be enormous. Size limiting factors are namely only node addressing and single hop length. The transmission distance between two nodes can be up to 100 meters open air, but a more realistic value taking environment variables and power efficient transmission powers into consideration lies somewhere

between 10 to 20 meters [Far08]. If node density is not an issue in a large sensor field, then addressing might put a theoretical cap on the networks geographical size. One coordinator node in a ZigBee network is capable of addressing up to $2^{16} - 1$ sensor nodes. If that amount is still not sufficient, then multiple coordinators can be linked to create an even larger network. However, in such a setup, there would be multiple meshes next to each other instead of only one mesh. In a large mesh transmission delay would in most cases not be a problem, since idle ZigBee nodes can wake up in down to 15 ms.

Another trivial example of an active thing is people. Most of us carry a personal mobile smartphone that is connected to a mobile network and is capable of providing data such as location for example. Because of personal and privacy reasons this raises doubts whether or not it is safe and smart to report your own location. However, many social applications already today provide such features. With location aware social applications people can be considered as things in the Internet.

2.2 Connectivity layer

In order to make the things described in Section 2.1 contribute to IoT they require some connectivity. This is ultimately challenging since the amount of connected devices is growing rapidly. According to an estimate [Eri11] there will be over 50 billion devices connected to the Internet by year 2020. In a network of that size it is crucial that device connectivity is scalable and robust.

The estimate of 50 billion devices is divided among all sorts of devices, such as personal computers, smartphones, tablets, audio equipment, televisions, various sensors and even vehicles. Some of these devices by nature have a static location and they can be connected to the infrastructure network through WiFi or cabling, while mobile devices have connectivity through IP based technologies such as LTE. Addressing of these devices should neither be a problem since the next generation Internet protocol, IPv6 [DH98], is capable of addressing up to $2^{128} - 1$ devices. To give some perspective to the numbers, IPv6 has an address pool of roughly $6.8 * 10^{27}$ times the estimated 50 billion devices.

Connecting various sensor devices as *things* to the Internet is challenging. Many of the WSN technologies that have emerged during the last decade, including ZigBee, have implemented their own protocol stack upwards from the data link layer. Reason for this is most likely the fact that physical layer standards suitable for sensor devices exist, but reckoned network layer standards tailored for sensor communication have not existed. Until recently a strong candidate has emerged.

The ideal in a worldwide sensor network would be to take the dominating Internet protocol (IP) all the way to the sensors without having intermediate translation layers or gateways. *IPv6 over Low power Wireless Personal Area Networks* [Mul07] (6LoWPAN) is a protocol especially designed for such scenarios. It is designed to take IP to the very edge of the network, including low power devices. How 6LoWPAN does this is that it carries IPv6 datagrams over IEEE 802.15.4 based networks, including ZigBee. Originally it was aimed to be an adaption layer to be able to transport IPv6 headers over any kind of medium. It has since evolved into mainly constrained and low power networks and was also warmly welcomed [SB11] by the Internet of Things.

6LoWPAN is a standard by the Internet Engineering Task Force (IETF). Design focus of the protocol is to be small. The conventional IP stack is not particularly big, but 6LoWPAN is even smaller. Its code size is even less than corresponding protocol stack code of ZigBee. Despite that, it is capable of addressing orders of magnitude larger networks – up to 2^{64} nodes. It even requires less RAM from the hardware for running the protocol. Even though it requires less of everything than similar protocols on top of the IEEE 802.15.4, 6LoWPAN uses well known UDP and TCP datagrams for messaging.

How 6LoWPAN manages to be so lightweight is through implementing stacked headers familiar from IPv6. Many packet based protocols, including IPv4 and ZigBee, uses one monolithic static size header. IPv6, as well as 6LoWPAN implements several types of headers to be used for different types of messaging. 6LoWPAN defines four types of headers: dispatch header, mesh header, fragmentation header, and the compression header. The trick with stacked headers is to send only the headers that are required. If for example the node is in a non-mesh network, it does not send a

mesh header. Or if its datagram is so small that it requires no fragmentation, the fragmentation header is left out. Only the payload carrying dispatch header and a compressed IPv6 header are the bare minimum to send in a trivial case.

One essential, and truly valuable, feature to keep in mind regarding 6LoWPAN is that it does not require any dedicated or proprietary hardware. No dedicated gateways, proxies or translation layers are required because 6LoWPAN datagrams are compatible with the existing Internet routers. This meets the requirements in the ideal of a worldwide sensor network where network addressing is flat and the same protocol is used on every edge.

2.3 Application protocol layer

As soon as the *things* have connectivity their data is ready to be propagated into the network. Various protocols exist for this purpose. Some of these protocols require at least UDP/IP connectivity from the sensors, while other are capable of communicating directly with the MAC layer of a IEEE 802.15.4 stack. There are also some technologies, such as ZigBee, have implemented their own protocol. The ZigBee protocol stack communicates straight with the 802.15.4 layer, but they are migrating currently to operate over 6LoWPAN [Stu09].

Next, we give a brief introduction to three technology independent application protocols designed for small data transmission from constrained networks, such as WSNs.

CoAP

Constrained Application Protocol [SHB13] (CoAP) is a widely adopted protocol in delivering sensor data over the infrastructure network. Its design goals are in simplicity and low overhead in order to make it suitable for resource constrained devices, such as low-power sensors. It has been designed to be so small and modest in terms of hardware, that it can be taken all the way to the sensor device with minimal calculation power. An operating system, such as TinyOS, and some sort of IP connectivity, 6LoWPAN or regular IPv6 for example, are required in order to make the sensor collaborate independently. In such scenarios a dedicated gateway is not necessary, which makes it a reckoned

protocol for distant and scattered singleton sensors.

CoAP is a HTTP counterpart, which means that it is possible to adapt on almost any UDP capable device. Due to an intentional HTTP impersonating design, it also integrates by default with the current Web. In addition to regular HTTP, CoAP implements some extra features, such as multicast, tailored specifically for sensor environments.

MqTT-S

Message queue Telemetry Transport for sensor networks [HTSC08] (MqTT-S) is another widely adopted protocol in the IoT field. It is a data centric publish/subscribe system that uses servers as message brokers for collecting, storing and distributing data. In a publish/subscribe model the sensor nodes, or the responsive sensor gateway assigns a topic for the sensor data. This data is then forwarded over IP to a message broker which stores it and looks up in a database for clients that have subscribed to data published under the corresponding topic. The broker server then pushes the data to all those entities that have issued a subscription to that specific data.

MqTT, from which MqTT-S is derived, is a in fact a messaging protocol used by several popular instant messengers. MqTT-S is based on same principles, but it has been designed extend connectivity beyond IP networks and to be more sparing in constrained M2M communication [SCT]. MqTT-S is capable of communicating directly with a IEEE 802.15.4 MAC layer.

STMP

Sensor data Transmission and Management Protocol [AF11] (STMP) is a transport framework designed for sensor data delivery. Sensor networks utilize various transport layer protocols for data dissemination, such as UDP, TCP, RTP [SCFJ03] and ATCP [LS01]. SMTP has taken this into consideration in its design by being flexible and capable to choosing the correct and most suitable protocol for the sensors. It is also designed to have minimal overhead since the targeted devices are in many cases power and network constrained.

Similarly to MqTT-S, STMP uses fixed services in the infrastructure network to collect data. These points are referred to as *fusion points*. Sensor devices

registered themselves to these fusion points by some transport protocol they support. End-user applications then connect through the fusion points to the sensor devices. This way the fusion point has all responsibility over trying to keep a reliable connection to the sensor, while the client only has to maintain a connection to the fusion point.

In addition to technology independent transport level data dissemination protocols for sensors, there are several technology specific and monolithic bottom-to-top stacks that support various application layer features. Probably the most notable one of this kind is DASH7 [Nor09] with their open source stack called OpenTag.

2.4 Semantic layer

The topmost layer in our IoT stack is the semantic layer. It is the highest abstraction level of IoT, which in the common case is also the layer that implements various interfaces for end-users. The essence of the semantic layer is to hide sensor accessing details from the user and provide some user-friendly application that use the underlying sensor networks. Services on the semantic layer can be considered as consumers of the information gathered by the content producers, or *things*, as we call them.

Real life applications that use sensor readings or triggers and control actuator devices need to be abstracted to the end-user in order to be effective and convenient to use. Something as simple as automatic lights at home can be complicated to configure without service abstraction for creating profiles that connect motion sensors with light triggers. Such services implement an *application programming interface* (API) to the sensors and actuators, which end-user oriented applications can use for simplified access to *things*.

Semantic layer applications usually follow a *service-oriented architecture* (SOA). How the underlying network of *things* is divided into service units depends heavily on the application. In other words, a service unit can either be a single device or a large set of *things*. There are several ways to implement service-oriented applications. One common way is to follow existing architecture styles, such as *Representational*

state transfer [FT02] (REST), which is an abstraction architecture for the Web. Its goal is to simplify remote access to resources, such as sensor devices. REST uses primarily HTTP as its application level transport protocol. It has even been defined in the context of HTTP, but despite that it does not require HTTP. Any protocol that provides a sufficient set of messaging methods is enough to provide the building blocks for a REST API. With the help of a REST API access to sensors can be simplified. Figure 3 illustrates a common scenario where a client uses a sensor device through a REST API without having to know anything about how to access the sensor on the protocol level.

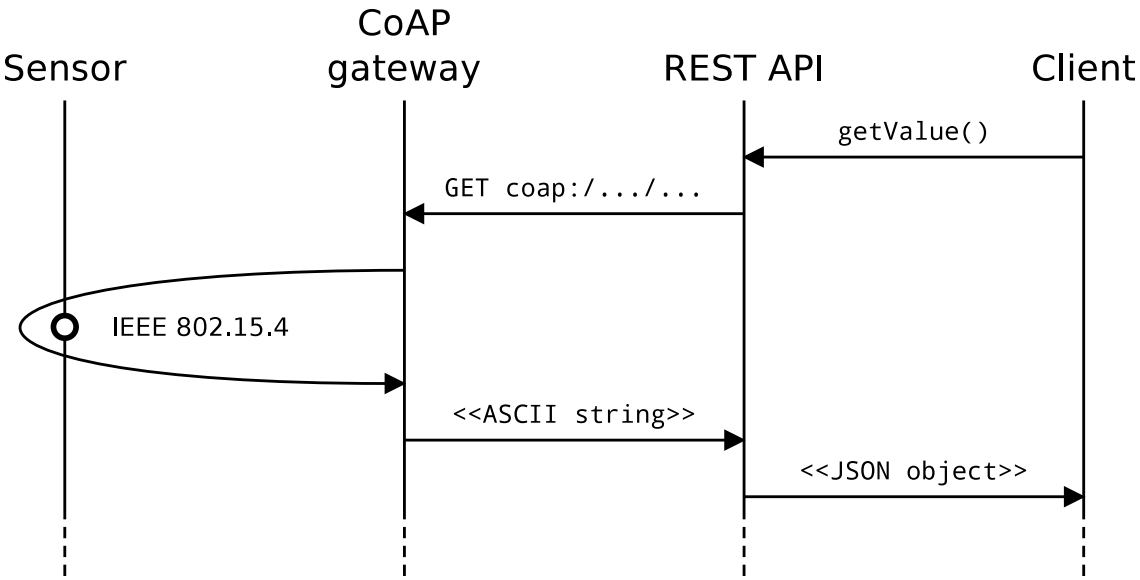

Figure 3: Client accessing a sensor through REST.

Another essential field of interest on the semantic layer is presentation of data. Large amounts of sensor data require some sort of visualization or representation in order to be easily understandable. Large and complex sensor fields for monitoring phenomenon like traffic, weather conditions and natural disasters produce lots of raw data. In order to make data directly useful for a human observer it has to be parsed for events of interest. This is achieved through semantic applications that process the data and draw conclusions based on it.

3 Information-Centric Networking

The design foundations of the current Internet architecture dates back several decades. Originally the motivation for networking was sharing of scarce and expensive resources, such as card punchers or mainframe computers. At that time the network's task was purely to deliver packets between two endpoints, which resulted in communication that was host-to-host by nature. This connection-centric paradigm met the requirements of networking at that time, and since then it has been generally acknowledged and growing continuously.

Roughly four decades later the Internet has evolved into something that was impossible to anticipate in its early days. Today connected hosts are more than the originally plentiful address pool can handle [ICA11]. The amount of hosted data is hard to estimate, but one thing for sure is that it is still continuously growing. Internet connectivity is being expanded to a wider scope of devices, such as smartphones, televisions, audio equipment and even vehicles. The worldwide network has grown enormously in every aspect, but the networking paradigm is still based on the original design.

During the evolution of today's Internet the common usage model has changed dramatically. Concepts like content distribution, mobility and security, which are probably the most desired properties today, were unknown in the early days. Various patches and protocols have been addressed to add lacking functionality to the Internet, but many of them have turned out to only complicate the overall architecture, and therefore sooner or later vanished from the network [Han06]. While the old fashioned Internet design struggles to keep up with the demands of today, a new networking paradigm, *information-centric networking* [XVS+13] (ICN), has emerged from the research community.

In the current Internet content is addressed by hostnames and paths. A hostname, which after resolving refers to an IP address, belongs to a device in the network. Therefore it is eventually pointing to a location. This architecture enforced strong coupling between data and location is in most cases unnecessary since the data does not have to be coupled to the location. ICN's primary purpose is to break this coupling through addressing data by its name rather than location.

Early publish/subscribe [EFGK03] systems in the 1990s can be considered as the first steps towards an information-centric networking model. Such systems are based on the notion of *topics* or *subjects*, that clients can either *publish* or *subscribe* to. While equivalents to these actions play a significant role in today's ICN approaches, the design of recent ICN implementations goes much deeper into network transport and routing mechanisms. Early publish/subscribe systems were purely application level implementations running on top of conventional IP.

In the recent years several different proposals of ICN architectures have been presented. The research communities' interest in ICN has increased after a much attention gathered Google Tech Talk [Jac06] held in 2006. The first complete ICN architecture, *data-oriented network architecture* [KCC+07] (DONA), which revolutionized the content addressing by replacing hierarchical URLs with flat names, was introduced in 2007. Since then other notable ICN architectures, such as *Named Data Networking* [NDN] (NDN), *Publish Subscribe Internet Technology* [PUR] (PURSUIT, follower of PSIRP), *Network of Information* [Net] (NetInf), *Content Mediator architecture for Content-aware Networks* [COM] (COMET) and *Convergence* [CON] have been introduced.

3.1 ICN fundamentals

Several different *Information-Centric Networking* (ICN) architectures have emerged during the last decade. Most of them, however, share same principles and characteristics. All of them have a common goal in trying to provide an alternative networking paradigm that would fulfill the requirements of today's Internet more effectively. In other words, and as the name claims, they all focus on moving the Internet away from the current connection-centric, or "client-server", model to a more suitable information-centric Future Internet [PPJ11]. In this Chapter we will look at design commonalities in different ICN architectures.

The basic building block in every ICN architecture is content addressing by its name. Current ICN architectures implement this in different ways. Some, such as DONA and PURSUIT, use a flat naming scheme for content. Information addressed with flat names must globally unique, which leads to non-human readable names.

However, unique names are self-certifying, because each name can point to only one unique data object. Therefore, the name of a data container can be used to verify that the integrity of the actual payload. A flat naming space has also advantages in mobility, since clients are unable to move from one domain to another due to the absence of the whole concept of domains.

In terms of scalability a flat naming space does not cope very well. Due to the lack of hierarchy, a flat namespace requires costly content resolution mechanisms for routing [XVS+13]. A hierarchical naming scheme provides a location-identity binding, which can be used to define routes for certain content in the network. While this binding is beneficial in scalability, it must be noted that defining the network topology should only be done on a sufficiently high level. After all, it basically is about binding locations to identities, which has been identified as one of the deficiencies in today's Internet architecture.

Name resolution is an open question in ICN. Majority of the ICN proposals, such as DONA, PURSUIT, NetInf and COMET, are based on separate resolution services in the network. Despite the fact that each one of them play a similar role in the network, they have been given different names; *Resolution Handler* (RH), *Rendezvous Node* (RN), *Name Resolution System* (NRS) and *Content Resolution System* (CRS) respectively. Separate resolution services deployed in the network resemble an awful lot the current DNS system, which has also been pointed out as a deficiency in today's Internet. In contrast, NDN and Convergence implement name resolution as an embedded feature in network routers. NDN does name resolution on each router on every object that it gets a request for. Content routers check their own cache initially on every incoming interest. If the router is capable of providing the requested data, it sends the data to the requester. If the content router does not have the data, it performs longest prefix matching on the hierarchical name, and sends it accordingly to the next router. Convergence implements a similar resolution scheme, but in addition to NDN's scheme Convergence has a fallback option of querying a separate resolution service.

Transparent caching of information is fundamental in ICN. A huge share of the information moving in the current Internet is being retransmitted from its original

host over and over again. In an ICN architecture where location has no value regarding the information, this kind of content awareness combined with caching could reduce the amount of traffic significantly. Caching and content awareness has a key property in information-centricness. All of the aforementioned ICN architecture proposals implement native on-path caching. On-path caching is an opportunistic caching strategy, where each router located on the designated data path caches everything that passes it. If there is a subsequent request for the same data crossing that path before the cached copy expires, the request can be satisfied with the cached copy without propagating the request any further.

Mobility is increasingly gaining more attention since Internet connectivity is making its way to all kinds of mobile devices. In the current connection-centric Internet mobility has been a primary shortcoming, since a connection hand-off is practically just shutting down one link and opening another one. Applications using a link hardly know how to react while disconnecting, and the outcome is many times no more than a humble apology to the user. This is a problem of stateful connections, such as TCP/IP. In contrast, ICN architectures use stateless connections. In case a client faces a connection hand-off by moving to another physical location, a new connection is likely to be established as soon as possible. If the spontaneous hand-off interrupted the transmission of an information block, a request for that same block should be reissued via the new link. Due to the stateless connections there is no handshake overhead by the protocol. Additionally, since the user cannot move long distances at once, probably only to an adjacent domain, the content he received only half-ways relies very likely on a geographically nearby router. ICN provides natively sufficient ways to handle receiver mobility. However, sender mobility is still an open question.

Trust and security is an open issue in ICN. In the current Internet architecture many trust and security models are based on authenticated hosts. In other words, data can be trusted if it comes from a source that can be proven to be legitimate. In an ICN architecture this kind of entity based trust cannot exist since valid information is designed to come from anyone. Therefore, security in ICN must be embedded into the information objects. This has been implemented in most of the aforementioned ICN architectures. However, security still remains as an open question.

3.2 CCN

The *Content Centric Networking* [JST+09] (CCN) architecture is a ICN implementation from Palo Alto research center (PARC). CCN is continuation to the aforementioned NDN architecture, which principles were initially presented in a Google Tech talk [Jac06]. The CCN project was chosen to be one of the supported projects by Future Internet Architecture program (FIA). FIA's purpose is to fund research projects regarding all kinds of design proposals for the future Internet. Four ICN architectures were chosen, including CCN as one of them. Recently CCN has gained a lot of interest since it is being actively developed and is a promising pioneering ICN implementation. The open source implementation of CCN is called CCNx [CCN].

CCN's primary task is replacing *where* with *what* in networking. As with most other ICN architectures, CCN addresses content by name, rather than location. In CCN this is achieved through elevating the so-called narrow waist of networking. Figure 4 illustrates the difference between current IP's and CCN's narrow waist. Elevating the narrow waist of the Internet architecture to the content layer and introducing a strategy layer between the content layer and the underlying network enables new ways of moving data. The transport medium can be practically anything, ranging from common IP to unreliable opportunistic networks, and even portable drives.

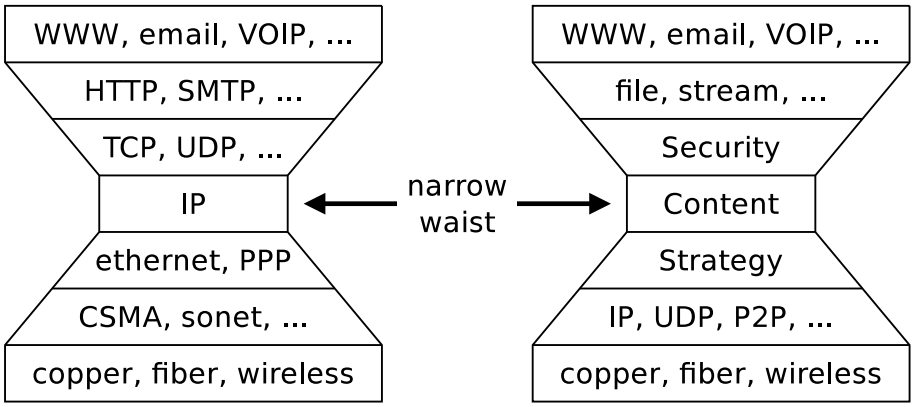

Figure 4: Current Internet architecture's networking stack (left) compared to the elevated narrow waist stack introduced by CCN (right).

CCN uses hierarchical naming of content. Its naming scheme resemble much the URL scheme of today's Internet. However, the big difference is that these names

are not pointing to locations. Valid names in CCN are in fact prefixes that are matched to existing content. As an example, a client requesting content by the name `ccnx:/foobar` could be satisfied by content that has been published by the name `ccnx:/foobar/index.html`. Since requests for content are issued on a prefix basis, the content provider, in our case foobar, would decide over what content to provide by default to a request with no further content description. After providing the default content, `index.html`, our example client might find inside that object a link to, say, `ccnx:/foobar/login.html`, which would address further content on the site he is browsing. The same analogy applies to dynamic content that is split into chunks and generated continuously, such as a live stream. A stream's handle could be obtained by the prefix `ccnx:/foobar/video`, which would provide the most recent chunk of the stream. Successive chunk names could be deduced from the received content, and by concatenating suffixes to the name prefix describe the whole chain of chunks, e.g. `/_c1`, `/_c2`, and so on.

Content in CCN is requested through issuing an *InterestMessage* (IM) describing a name prefix. IMs are satisfied with content containers known as *ContentObjects* (CO). Issued IMs are routed with a hop-by-hop basis through *Content Routers* (CR). Content routers consist of three main data structures, FIB, PIT and CS. These data structures are explained below.

Forwarding Information Base (FIB)

FIB is a data structure that contains forwarding information that is used while routing IMs. Entries in the FIB are pairs of specified name prefixes and outgoing faces. The CR does longest prefix matching on incoming IMs, and based on the outcome decides which outgoing face potentially leads to the requested data.

Pending Interest Table (PIT)

PIT keeps track of every IM that have been forwarded by the CR. Each passing IM leaves an entry in the PIT. These entries are used if the IM is eventually satisfied and the content has to find its way back to the original issuer of the IM. PIT entries are analogous to breadcrumbs, which are used to keep track of a traversed path.

Content Store (CS)

CS is the cache of a CR. By default every CR does caching through storing passing COs in the CS. Every CR checks their CS on each incoming IM before deciding whether to forward it or not. If the CR is capable of providing a cached copy of the requested data it is prioritized over propagating the IM further to the network.

In addition to the CS, CCN also provides repositories for persistent storage of content. These repositories can be separate services deployed in the network, or alternatively running within a content router. Content has a limited lifetime in routers. COs can be generated with a lifetime parameter, *FreshnessSeconds*, which indicates how long it will take for the content to become stale. If a CS' capacity is about to get filled, stale data is primarily purged from the CS. If the CS is about to get filled with non-stale data, *fresh* data is removed on a *least-recently-used* (LRU) or *least-frequently-used* (LFU) basis. Therefore, there is no guarantee over how long data will remain in the CS. CCN repositories are designated to provide long term storage for content.

Security, as earlier mentioned in Section 3.1, has to be embedded within the content. In CCN all public content is authenticated with digital signatures. Private content on the other hand is encryptable with encryption keys, which the entities have to exchange by themselves. In order to make public content authentication less troublesome, *ContentObjects* have designated fields for embedding a key within the content or to carry a key locator that provides the authenticator a name by which the correct key is retrievable. This public content authentication is purely syntactic, since malicious content can be equipped with a key that authenticates the bad content. In other words, content may be authenticatable, but that does not mean its valid or trustworthy data. However, the key issuer can of course be validated too. As earlier mentioned in Section 3.1, security in ICN architectures is still an open question. A built-in authentication mechanism in CCN is a building block for a robust way of authenticating data, which is yet to come.

4 CCN in a sensor environment

This thesis work combines *Content-Centric Networking* (CCN) and the concept of *Internet of Things* (IoT). In this Chapter we look at different ways on how to make use of the hierarchical CCN-content naming, in-network caching and other information-centric networking characteristics in a sensor environment.

4.1 Motivation

Many of today's wireless sensor networks (WSN) are completely segregated and isolated. In critical applications that is intentional and desired in terms of security and safety. In contrast, many non-critical sensor networks could contribute, or benefit from being connected to a bigger ecosystem. Extensive connectivity of the sensor network nodes is required in order to participate in a larger ecosystem, such as the IoT introduced in Chapter 2. However, extensive connectivity is not that easily achieved due to hardware limitations and several different competing technologies.

Current WSNs are built using various different technologies. These technologies provide a wide selection of characteristics to suit almost any kind of sensor network. Some WSN applications might find constrained and energy efficient operation most important, while some other sensing environment may depend on long distance connectivity. Most of these technologies operate on protocol specific hardware due to dedicated frequencies and sensing network topologies.

Due to the differences in sensor network hardware most of the technologies are incompatible with each other. There is a standard [43407] which aims at unifying sensor to host communication. This standard covers all IEEE 1451.5 approved technologies, namely IEEE 802.11 [SCC13], IEEE 802.15.4 [CGH+02], Bluetooth [LDB03] and ZigBee [Far08]. The standard specifies communication over the air from the *Wireless Transducer Interface Module* (WTIM) to the next *Network-Capable Application Processor* (NCAP). It also specifies communication between interconnected NCAPs, but it does not take account to how the data should be further propagated to the network from the NCAPs. This is intentional since the standard's scope is only between the physical and transport layers.

Standardizing and unifying sensor technologies is a step in the right direction towards extensive connectivity of sensors. However, there is no standard that would specify how sensor data should be handled on the application layer. Currently there are several IoT-oriented protocols competing. Some of them are introduced in Chapter 2.3.

Many of the IoT-oriented protocols, such as CoAP and REST, are HTTP counterparts. It means that they operate on top of an IP stack, which requires that each contributing device has to be uniquely addressable through underlying IP mechanisms. HTTP by specification does not require a certain transport and network protocol pair, but TCP/IP is by far the most commonly used. Due to old fashioned principles of IP we may run into problems concerning scalability and address allocation in some environments.

Probably the biggest concern right now is address allocation. According to an estimate [Eri11] there will be over 50 billion devices connected to the Internet by year 2020. That is over 11 times more devices than the traditional IPv4 theoretically is capable of providing unique addresses to. Due to sloppy allocation the available addresses are even less in practice. Therefore it is quite obvious that IPv4 is incapable of providing connectivity to all potential devices.

As a solution to IPv4 address space shortage the next version, IPv6, has been introduced. Even though IPv6 has been around since the late 1990s, it has not seen a wide scale breakthrough in usage yet. Reason for its slow deployment is explainable through *Network Address Translation* (NAT), a workaround to escape the address shortage, and IPv6 tunneling through IPv4 networks, allowing the non-interoperable IP versions work side by side. Statistics by Google [Goo] show that their user activity over IPv6 reached a one percent share in November 2012.

Despite the fact that IPv6 would provide plenty of addresses for future devices and its use is continuously growing, all connections would still be point-to-point. That is not necessarily a bad thing, but in our solution we are looking at a different approach. Point-to-point connections work fine when both entities are static and singletons. If either of the entities are mobile we face a mobility problem. If, on the other hand, the receiver entity is a set of individual recipients we have to either rely

on multicast or deploy a message broker, proxy server or a gateway in the network.

Deploying extra services to the network is not the most elegant way to solve a one-to-many problem. Message brokers and proxy servers introduce a single point of failure (SPOF) in the network unless replicated to an adequate degree. Gateways introduce a SPOF as well, but depending on the topology its downtime could make the sensors behind it inaccessible as well. A demand for separate services deployed in the network would break the seamless interoperability the sensors could otherwise have.

Multicast, on the other hand, is not guaranteed to be supported in every network. Besides that, multicast is sender driven. This would mean that every data object would be necessary to send whether or not anyone would need it at that time. Also, sensor data granularity would thus depend on transmission frequency of the sensor, which could cause excessive power consumption of a low power sensor device.

We try to tackle these issues by presenting CCN as a possible transport protocol for sensor data. Most importantly, CCN is designed to be independent from transport and network layer protocols. Data in a content-centric network is addressed by the actual content instead of where the data is hosted. Therefore interconnection between CCN nodes does not rely on any addressing scheme from a network protocol, such as IP. However, because CCN nodes are not yet widely deployed, it is capable of operating on top of IP for the time being.

CCN also benefits from simultaneous connectivity models since it uses stateless connections. It has weak demands on the data layer (OSI layer 2), which makes it good for unreliable connectivity. In a worst case scenario data objects from a sensor device could be transparently delivered to the network through opportunistic network technologies, or any other CCN supported type of moving data.

With CCN there is no need for message broker servers or proxies. A message broker server in general is needed when there has to be some centralized system that collects data from the sources and delivers it to all the sinks with subscriptions to the data in question. In CCN such message brokers are not needed because all the clients can subscribe to any data they are eligible to get through issuing an *InterestMessage* describing the data. Since content-centric networking does not address content by

location, the data is either at the source node or located on a path leading to the source. It might even be at some of our client's neighbors, in case it had retrieved the same data object prior to our client.

An intermediate proxy is another way to enable client-to-sensor connections. There are usually other motivational factors as well as simply enabling connections. Proxies usually provide caching of data in order to reduce the workload of the data source. A proxy may also be deployed if the data sources are in a private network which is accessible only by the proxy from the outside. Caching is done by default on all CCN enabled routers. In other words separate proxies are not needed in a CCN network, since all the routers take care of the caching. In a private network scenario some extra service is needed. A dedicated CCN repository outside the private network gathering all data objects from the sensors would be one way to work around the limited access network. On the other hand, if network addressing was flat there would not be unintentional private networks accessible only through proxies or NAT technologies.

In our approach we aim for a higher abstraction level in accessing services and sensors, or so-called *things* in the Internet. This higher abstraction is achieved through CCN's hierarchical and descriptive addressing of data. Another key interest for bringing CCN to IoT is in-network storage and caching of content. Caching in CCN is built in and it is done by default on every CCN-enabled router. Content-centric networking suits our vision well, which we will explain and demonstrate in the following chapters.

4.2 One-time data retrieval

In CCN data exchange is always pull-driven. Data transmission is initialized with issuing an *InterestMessage* (IM) describing the wanted data. The IM is generated based on the descriptive and hierarchical name of the requested data. It may also contain additional bits of information describing the data. This issued IM is propagated in the network according to the specified routing strategy and each node's *Forwarding Information Base* (FIB) entries. It is being passed around the network until a node has data that satisfies the IM or its *Interest Lifetime* expires.

While the IM advances in the network it leaves an entry in every CCN routers' *Pending Interest Table* (PIT) that it passes. When an IM eventually reaches a node that holds the requested data, the data is sent back the same route following PIT entries left on the path. Intermediate routers does not only forward data but also store a copy of the passing data. This is a built in feature in CCN to provide transparent in-network caching.

Many low-powered sensor devices can benefit from letting the network store and further propagate sensor data. Consider a scenario in which n clients scattered around a network are interested in sensor data d generated at a specific time t. Let us denote this data object by $d(t)$. Each one of the n clients generates an IM that matches $d(t)$. The IM's are dispatched approximately at the same time. Due to network latency and other transport variables we cannot say which one arrives first. One of them arrives first at the sensor or its closest responsible CCN-enabled gateway. The first arriving IM, denoted by i_0, is replied with a generated *ContentObject* (CO) containing the requested data $d(t)$. This newly generated CO is delivered back to the issuer of i_0 and the same path in reverse order following PIT entries left on the path. On each router on the path a copy of $d(t)$ is left to provide the same data for possible future interests. If one of the remaining IM's (i_1, ... , i_n) happen to pass one of these routers they are satisfied with the cached copy of $d(t)$.

The *FreshnessSeconds* of $d(t)$ should be set to match the measuring time granularity of the sensor device. Some sensor devices report updated values only between a

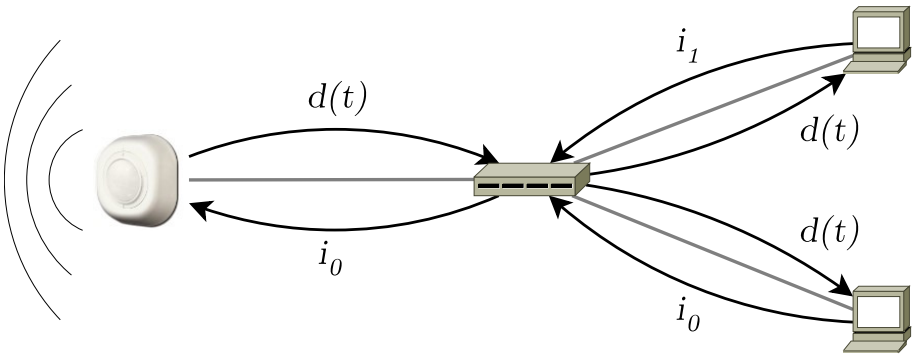

Figure 5: Two clients are interested in data object $d(t)$. Intermediate CCN router provides a cached copy of $d(t)$ in exchange to the second interest i_1.

specified time interval of up to several minutes. In such cases it is unnecessary to dispatch multiple instances of COs with exactly the same data. Figure 5 illustrates one-time data retrieval from a sensor device.

4.3 Stored data retrieval

Caches, or *Content Stores* (CS) in CCN context, introduced by every CCN router are not a persistent data store. There are no guarantees of how long a CO will stay in a CCN cache. In order to store long-term and historical data in the network we have to establish a CCN repository on some router. The repository can be configured to store all COs that passes by it and satisfies criterias regarding the data we want to store. If, for example, we want to store readings from `ccnx:/alice/home/temperature`, we could define every CO that matches that name prefix to be taken into the repository.

An alternative way to push data into a repository is through issuing a *Start Write* (SW) command from the sensor side to the repository. After a successful SW command the repository requests for data described in the SW. This way the sensor can fully control all the data that it puts to the repository. It does not have to count on that its COs pass the responsible repository.

Let us take an alkaline battery powered household-oriented temperature sensor for example. Due to power saving behavior it might report its reading only every couple of minutes. Instead of dispatching its data to the CCN network only on demand

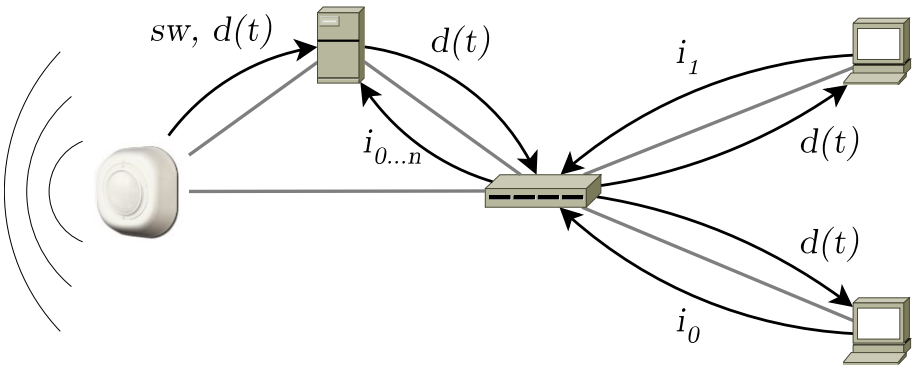

Figure 6: Sensor node pushes its data to a CCN repository. Data is available at the repository even if cached copies at the CCN router had expired.

it could push the value of each measuring point to a repository. The repository is capable of storing historical data and serving all incoming IMs targeted to the sensor. Figure 6 illustrates a scenario where a repository is used to aggregate and propagate sensor data.

Whichever way is utilized to propagate data into the repository, once it is there it is persistent. Clients that request the stored data issue a normal IM and in return they will get a matching CO from the repository. This leaves the sensor intact reducing its workload.

4.4 Actuators

Remotely controllable things in the Internet provide telecommand features. Such features require actuator commands to operate. An actuator command targeted to a specific device contains information about which action to perform. For example, possible actuator commands for a remotely controllable light bulb could be state changes between on and off, or a percentage to dim the light to.

As earlier mentioned, CCN is always pull oriented. Therefore, data containing the wanted state cannot be pushed to the remotely controllable device. Instead we can request for a certain state. Technically an actuator command is very similar to the one-time data retrieval explained in Section 4.2. The actuator message is constructed like any other IM. Instead of content this actuator IM shows interest in certain action.

Consider a scenario where a client, Alice, wants to switch her lights on. Alice generates an IM, which is addressed to the light switch. In order to make it an actuator message, a prefix describing the wanted action is appended to the name. For example, Alice could generate an IM addressing content by the name `ccnx:/alice/light/on`. Let r_0 denote this IM. A CO satisfying r_0 must not be available in the network. Like any other IM, r_0 is routed according to longest prefix matching and optional routing rules.

Eventually r_0 arrives at its destination. The longest prefix match is achieved as close as CCN is capable of going to the actuator device. It is now up to the device,

in this example the light switch, to initially parse the last name component to see which action is requested and to make sure that the issuer of r_0 is eligible to perform that action. In case both conditions are met the action can be carried out. Whether the outcome of the action was a success or not, Alice must be informed about it. According to the CCN protocol specification [CCN] an IM must be satisfied with a CO, or else the IM is considered as unsuccessful. The actuator device now generates some payload based on the outcome of the action, and wraps it in a CO and sends it back to Alice in return for r_0. We refer to this acknowledgment object as $a(r_0)$, which technically is a CO. This example is illustrated in Figure 7.

One important thing to note regarding the acknowledgment object is that its *FreshnessSeconds* must be set to zero. If it had a lifetime longer than zero, we would break the invariant regarding actuator commands not having matching data present in the network. In other words, we do not want to keep the acknowledgment objects alive in the CCN caches.

Using CCN for actuators as explained here is contradictionary to the philosophy of *Information-Centric Networking* (ICN). First of all, in case of a remotely controllable device the location of the device usually matters significantly. ICN tries to hide the source of the data, while with an actuator command in question the physical location of the source has to be unique, and in most cases also well known by the end user. Secondly, actuator commands do not benefit at all from in-network caching. In fact, caching of COs, which are used as acknowledgment messages, would be harmful to the operation. However, caching stale data is not harmful, but its persistency is not guaranteed. Instead of providing a perfect solution, our goal is to give a practical example of actuator commands over CCN as a proof on concept.

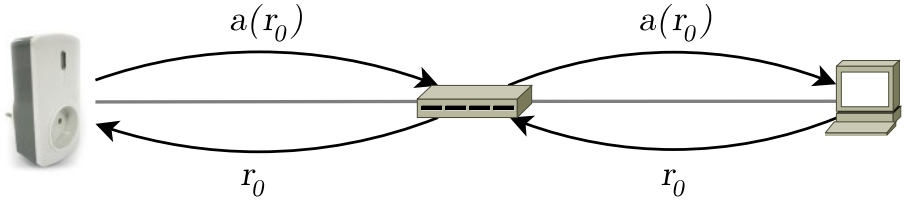

Figure 7: Client requests for an action. Successful execution of the request is acknowledged with a $a(r_o)$ message.

5 Testbed implementation

In order to evaluate the benefits of CCN in a sensor environment we implemented and established a testbed. The platform we used for the implementation is a home automation system provided by There Corporation. The automation system supports wireless sensors for various purposes, such as temperature, humidity and energy consumption measuring. Our testbed was deployed in a greenhouse located on the CS department roof. The greenhouse was originally founded for other research matters, but for our purpose it was a convenient place to gather actual data and possibly even attract public interest.

5.1 System overview

Backbone of our testbed implementation is a Linux-based router device called the ThereGate [The]. It is a commercially available home automation system provided by There Corporation. The ThereGate supports various wireless technologies, such as Z-Wave and ZigBee. Additional technologies' support can be added through USB and their corresponding drivers. Despite the variety of available technologies we will discuss only Z-Wave in this thesis. Same principles can, however, be applied to any technology.

The ThereGate uses DBus for internal communication between different software components, such as technology drivers and presentation bridges. A technology driver provides an application programming interface to the physical sensor devices, while a presentation bridge is an application interface to remote clients. Primary remote access method to the ThereGate is through HTTP. Figure 8 illustrates how machine-to-machine (M2M) connectivity is built between the network and the sensor devices through the ThereGate.

5.2 CCN presentation bridge

Our testbed implements a new presentation bridge for the ThereGate. It communicates to the external network through CCN. We call it `pb-ccnx`. The default

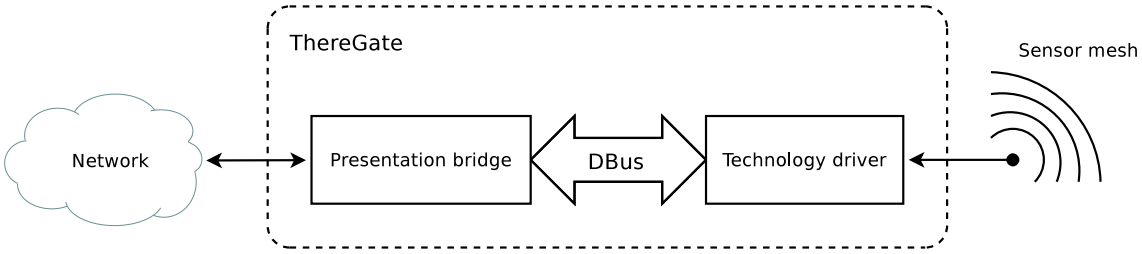

Figure 8: M2M connectivity through the ThereGate

HTTP presentation bridge works like any other HTTP client-server application; data is requested with `'GET'`, and if successful response is given back with a `'200 OK'` header. In CCN data exchange is different. Data is requested through issuing an *InterestMessage* (IM) describing the data, which is satisfied in return with a *ContentObject* (CO) containing the actual payload. Chapter 3.2 explains Content-Centric Networking more in detail.

On the ThereGate there is a CCN repository running for local storage of sensor data. The repository is capable of storing historical data practically as much as needed. Storage space is not a limiting factor as data containers are relatively small, and storage is always extendable through USB mass storage. As well as with any other data in a CCN repository, this data is also dispatchable to the external network if a client happens to request a past reading, or even a serie of consecutive readings for charts or diagrams.

5.2.1 Component description

The core of our CCN presentation bridge, `pb-ccnx`, is strongly coupled with CCN and ThereCore. It is also coupled to the CCN repository implementation, but it is not compulsory for `pb-ccnx` to operate. Both `ccnd`, daemon for CCN connectivity, and `ccnr`, CCN repository, are available as a part of the CCNx open source project [CCN]. ThereCore on the other hand is proprietary software owned by There Corporation. ThereCore is the core software on the ThereGate that communicates with the sensor hardware on the system. It provides a C API for all ThereCore features for implementing new technology drivers, presentation bridges and other application specific needs. Figure 9 illustrates how the components are connected

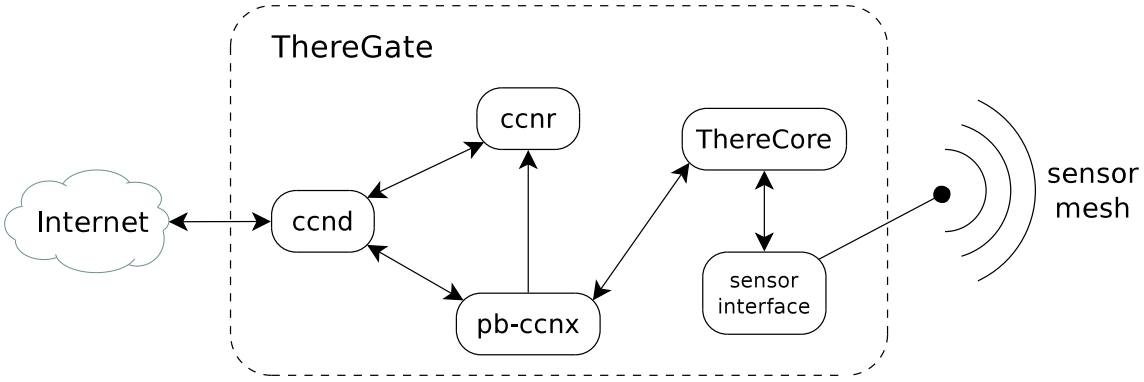

Figure 9: Visualization of the interconnected components and data flows.

to each other.

As figure 9 shows, our work connects these two sides. We release `pb-ccnx` source with this thesis, but it must be pointed out that it depends on ThereCore, and is most definitely not going to work on any other system as it is. Further implementation details on `pb-ccnx` are explained in Chapter 5.2.3.

An incoming IM can be treated in three different ways depending on the use case. The three use cases are explained below.

I. Interest for current reading

In this use case the client is interested in what the sensor reading is at the moment. Names described in such IM are registered by threads launched by `pb-ccnx`. There is a dedicated thread running for each sensor interface provided by ThereCore. This responsible thread sends a signal on the DBus requesting for the latest value. An example IM for this scenario could be issued for a name like `ccnx:/alice/home/temperature`. The IM is eventually satisfied with a CO generated and dispatched by the responsible thread. Chapter 4.2 explains a detailed example scenario of this use case.

II. Interest for historical reading

Persistent data is stored in a CCN repository. Therefore, in this use case the sensor specific handler thread is not responsible for generating, nor dispatching, the response CO. It is the repository's responsibility. Historical data objects can not be stored with an overlapping name. Thus we append Unix

timestamps to the content name in order to describe and granularize the data. The timestamps are also used in retrieval of sensor reading from a longer time span. Chapter 5.2.2 has a detailed description of the timestamp usage. An example of such a use case is presented in Chapter 4.3.

III. Interest as an actuator

This is similar to use case I. The difference is that instead of retrieving a value from the sensor device we request it to perform an action. The IM is issued with a name describing the action, such as `.../lights/on` for example. It is the handler thread's responsibility to signal the action request to the DBus, wait for a signal about the outcome, and finally satisfy the IM with a CO that will inform the client regarding the outcome. A more detailed description explained through an example can be read in Chapter 4.4.

There is also a fourth kind of scenario which differs in a fundamental way from the three cases explained above. It is the scenario where the sensing device is the trigger. As earlier mentioned CCN data exchange is always pull driven. Therefore, the sensor is unable to push data to the network. As initializing data transfer in the opposite direction is very contradictory with the whole concept of CCN, this has been left outside the scope of this thesis work.

5.2.2 Messaging format

Sensor readings are wrapped as JSON objects in CO. We decided to use JSON for data representation instead of XML because JSON is designed to be minimal and portable [Cro06]. It has a slightly simpler syntax and causes less markup overhead due to being a little more concise. The payload is wrapped into the CO as plain-text. If the data would have to be encrypted, CCN provides native mechanisms for encrypting the payload. Figure 10 shows an example JSON object carrying a temperature reading. It also shows other vital members. Table 1 explains the different members in a `pb-ccnx` message.

As CCN names are prefixes that match to content, we need to have a mechanism to address chains of COs in order to retrieve sensor data from a time span. We did this

Member	Description
ts	Contains a timestamp (ts) of the moment the current sensor reading was taken in Unix time format.
prev	Unix timestamp of the previous (prev) sensor reading.
data	This member contains an array of objects wrapping attribute (attr) and value (val) tuples. This is where the actual data travels. Objects in this array can be multiple, in case the sensing device provides various strongly related readings.

Table 1: JSON object member table

```
{
  "ts":"1379431971",
  "prev":"1379431671",
  "data": [
    {
      "attr":"Temperature",
      "val":"22.50"
    }
  ]
}
```

Figure 10: Sample JSON object containing a temperature reading.

by carrying the timestamp of the previously issued CO from the same data source. The timestamp of the previous CO is then used as a suffix after the name registered by the sensor in order to describe a CO from a certain time. This way after retrieving one CO from a sensor, we can use the previous timestamps iteratively to aggregate past COs just like a linked list. By default the registered name prefix returns the most recent CO. Figure 11 illustrates this linked list created by COs generated at different times.

5.2.3 Implementation details

Our presentation bridge implementation is written in ANSI C89. C was an obvious choice, because both CCNx and ThereCore are implemented in C. Both APIs are therefore convenient to use respectively. In addition to standard GNU/Linux

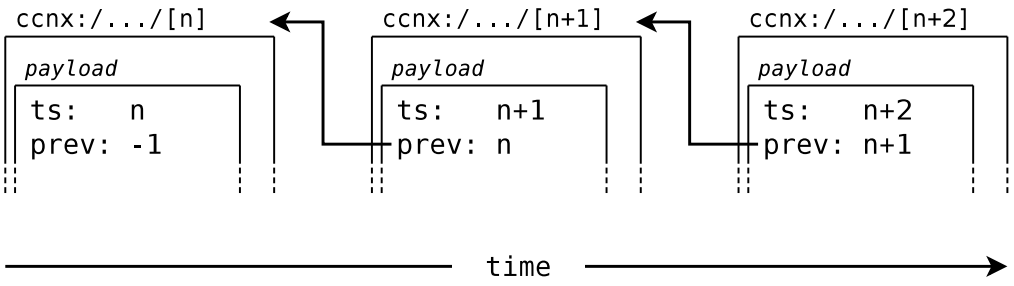

Figure 11: Linked list construction where previous link is carried within the payload.

libraries, our implementation depends on Posix thread support since it is heavily threaded. Other dependencies, such as OpenSSL and DBus, are implied by CCNx and ThereCore.

When `pb-ccnx` is initially launched it connects to the locally running CCN daemon, `ccnd`. After successfully connecting to `ccnd` it registers a predefined CCN name prefix for the sensors it is going to serve. This name prefix can also be considered as a path name common to all the sensors this current `pb-ccnx` instance possesses. These name prefixes follow the CCN descriptive and hierarchical naming conventions. To avoid confusion with terminology regarding CCN name prefixes and DBus paths, we will refer to the name prefix shared by all local sensors as their namespace. As an example, a valid namespace for a set of sensors could be `ccnx:/alice/home`.

Once `pb-ccnx` has registered its namespace it makes a DBus name request. A name is required for every client operating on the DBus. For this purpose we use `com.there.pb-ccnx`. Once `pb-ccnx` is connected to DBus, it queries for present sensor devices. ThereCore represents sensor devices on three levels; P-, L- and I-devices. We only care about I-devices for now. P-devices (physical) are used for lowest level access to the actual device. L-device (logical) layer is on top of the physical layer in case a single sensor device provides several different measures. All of these transducers providing readings can be seen as separate sensors with the help of L-device mapping. I-devices is the highest abstraction level of representing sensors. All present I-devices are iterated through and a thread is started within `pb-ccnx` to serve the sensor. We refer to these threads as handler threads from now on.

Each I-device has a human friendly name defined by the ThereGate configuration

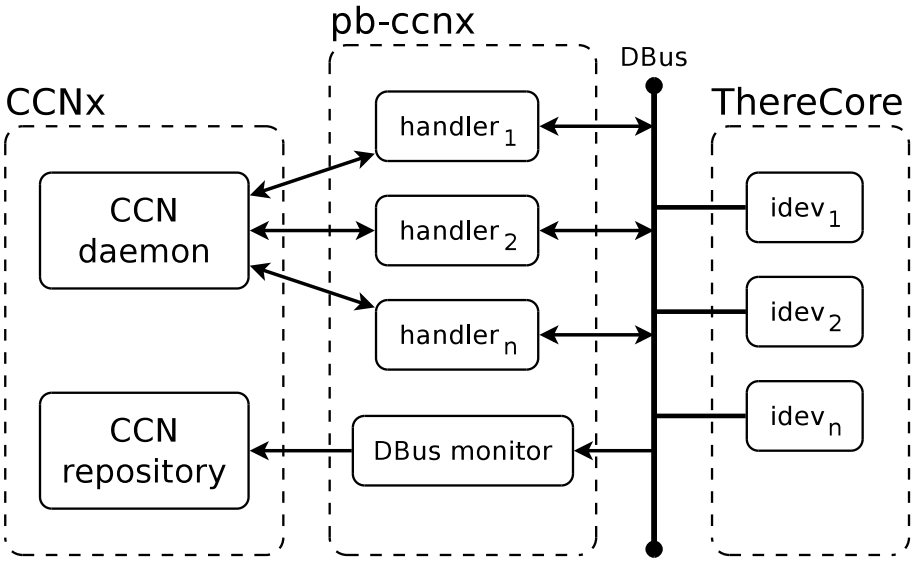

Figure 12: Illustration of software component communication on a thread level.

software. This name is user definable for every sensor and it can be any alphanumeric string. Each sensor's human friendly name is fetched from the ThereCore via DBus by `pb-ccnx`. Every handler thread registers then the top level namespace concatenated with its corresponding sensor name as its own name. As an example, the thread might register a name like `ccnx:/alice/home/temperature`. After `pb-ccnx` has iterated through all available I-devices, all of its sensor threads go into a stand by state. In the stand by state each thread is waiting for IM matching the name it earlier registered to.

In addition to the I-device specific handler threads `pb-ccnx` also runs a DBus monitor thread that waits for updates from the sensor side. All data retrieved this way is put into the repository, since there are no pending IMs for this data. Figure 12 shows a diagram of component connections on a thread-based communications level.

5.3 Experiment & evaluation

Our CCN presentation bridge was installed on a ThereGate that was coupled with temperature, humidity and energy consumption meters. The experiment testbed was deployed in a greenhouse established for other research purposes. Our CCN presentation bridge was accessible from the infrastructure network, but since CCN

routers are practically non-existent, we did not have any public interest in our sensor readings. The data was available for the public, but since we did not advertise the data anywhere, no CCN router could have located our content by other means than IM flooding through broadcast. The sensor data was not advertised since the nature and maturity of our implementation is still experimental. Also because this thesis work is a proof of concept rather than a finished product.

In this thesis work we have given a proof of concept that sensor data dissemination is possible over CCN. In addition to the proof of concept we stated explicitly some research goals in Section 1.2, which we sum up in the rest of this Chapter.

1. **No point-to-point connections**

   One of the basic architectural key points in *Information-Centric Networking* (ICN) is the absence of host notion. Therefore, this goal can be considered as achieved, since the communication paradigm in ICN is not based on host-to-host connectivity.

2. **Transparent in-network caching**

   On-path caching is done natively by all ICN architectures presented in Section 3.1. The end user does not have to explicitly know anything about cached copies, since the network is responsible of providing the most recent valid content. Therefore, this goal can be considered as achieved.

3. **In-network storage of sensor data**

   For persistent storage of data CCN provides native support for repositories, that can be used to store data. A repository does not exist by default anywhere, but it is supplied with the CCN basic distribution and it can be ran on any router or CCN capable device. Therefore, this goal can be considered as achieved.

4. **Reduced workload for the sensor devices.**

   On-path caching can reduce the workload of a sensors device dramatically, especially if the device attracts lots of interest. With the help of CCN nodes' caching features, this goal can be considered as achieved, since the network is capable of hosting content supplied by the sensor.

5. Provide a high-level abstraction layer to access sensor devices

A key design feature in ICN architectures is content addressing through names. Many architectures, CCN for example, provides hierarchical and human readable naming of content. Therefore, this goal can be considered as achieved since the the CCN naming scheme can be used to create high-level abstraction access to devices.

However, there are some notable drawbacks to take into account when considering CCN for a sensor environment. The fatality of these drawbacks depend much on the application. Our small scale testbed did not suffer notably from these drawbacks, since the our router had constant power and it had fixed Internet connection.

I. Overhead

Generating IM and CO, and especially signing them, requires some processing power. If compared to a simple HTTP variant, CCN wastes more resources in dispatching content. Also, a CO is slightly larger due to signatures embedded in the content for authentication purposes.

II. Moderate complexity

As the CCN protocol for transport is not as simple as HTTP, it requires some computational capabilities. In our implementation CCNx was running on a Linux based router device which has sufficient capabilities of simple cryptography capabilities for content signing. Executing the current CCN protocol stack on some less powerful devices, such as the simplest transducers with IP connectivity, is highly unlikely to happen because of the constraints regarding processing power.

6 Conclusions

Current Internet is undergoing some fundamental changes. The amount of connected devices is increasing rapidly and the whole field of networking is changing. The nature of connected devices is changing while more and more mobile devices, home electronics, sensors and even vehicles are equipped with Internet connectivity. In

Chapter 2 we presented the novel concept of the *Internet of Things* (IoT). In this concept the Internet evolves into a large sensing ecosystem, where *things* have a diffuse definition of being practically anything that influences with the real life and contributes to the IoT in order achieve common goals.

The increasing amount of connected devices is a major challenge to the current Internet architecture. To address this challenge research communities have proposed several *Information-Centric Networking* (ICN) architectures to replace the current networking paradigm to some extent. In Chapter 3 we presented ICN architecture fundamentals and one architecture implementation, *Content-Centric Networking* (CCN), in further detail. The current Internet architecture addresses content by location, which has lost much of its significance since most of today's Internet traffic is by no means coupled to location. In other words, ICN architectures urge to drive the communication paradigm from *where* to *what*, and address content by name instead of location.

We came up with the idea of combining ICN with the concept of IoT. We see that the IoT field could benefit from several ICN native properties. In Chapter 4 we introduce this combination, evaluate some benefits that could be achieved this way and eventually present how CCN would work in a IoT application on a practical level. As a proof of concept we implemented a CCN interface for a home automation system, that supports various sensor devices. This implementation is was presented in Chapter 5. In Section 1.2 we made some research goal statements. Whether we achieved the goals or not was evaluated in Section 5.3.

References

43407 IEEE Standard for a Smart Transducer Interface for Sensors and Actuators Wireless Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats. *IEEE Std 1451.5-2007*, pages C1–236.

AF11 Aschenbruck, N. and Fuchs, C., STMP — Sensor data transmission and management protocol. *Local Computer Networks (LCN), 2011 IEEE 36th Conference on*, 2011, pages 475–483.

AIM10 Atzori, L., Iera, A. and Morabito, G., The internet of things: A survey. *Computer Networks*, 54,15(2010), pages 2787–2805.

Ash09 Ashton, K., That 'Internet of Things' Thing, June 2009. URL `http://www.rfidjournal.com/articles/view?4986`.

BGS00 Bonnet, P., Gehrke, J. and Seshadri, P., Querying the physical world. *Personal Communications, IEEE*, 7,5(2000), pages 10–15.

CCN CCNx, *Content-Centric Networking*. URL `http://www.ccnx.org/`.

CGH+02 Callaway, E., Gorday, P., Hester, L., Gutierrez, J., Naeve, M., Heile, B. and Bahl, V., Home networking with ieee 802.15.4: a developing standard for low-rate wireless personal area networks. *Communications Magazine, IEEE*, 40,8(2002), pages 70–77.

COM COMET, Content mediator architecture for content-aware networks. URL `http://www.comet-project.org/`.

CON CONVERGENCE. URL `http://www.ict-convergence.eu/`.

Cro06 Crockford, D., The application/json Media Type for JavaScript Object Notation (JSON), RFC 4627 (Informational), July 2006. URL `http://www.ietf.org/rfc/rfc4627.txt`.

CWKS97 Crow, B., Widjaja, I., Kim, J. G. and Sakai, P., Ieee 802.11 wireless local area networks. *Communications Magazine, IEEE*, 35,9(1997), pages 116–126.

DDWL11 Durand, A., Droms, R., Woodyatt, J. and Lee, Y., Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion, RFC 6333 (Proposed Standard), August 2011. URL http://www.ietf.org/rfc/rfc6333.txt.

DH98 Deering, S. and Hinden, R., Internet Protocol, Version 6 (IPv6) Specification, RFC 2460 (Draft Standard), December 1998. URL http://www.ietf.org/rfc/rfc2460.txt. Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946.

EFGK03 Eugster, P. T., Felber, P. A., Guerraoui, R. and Kermarrec, A.-M., The many faces of publish/subscribe. *ACM Comput. Surv.*, 35,2(2003), pages 114–131. URL http://doi.acm.org/10.1145/857076.857078.

Eri11 Ericsson, More than 50 billion connected devices, February 2011. URL http://www.ericsson.com/res/docs/whitepapers/wp-50-billions.pdf.

Far08 Farahani, S., *ZigBee Wireless Networks and Transceivers*. Newnes, Newton, MA, USA, 2008.

FT02 Fielding, R. T. and Taylor, R. N., Principled design of the modern web architecture. *ACM Transactions on Internet Technology (TOIT)*, 2,2(2002), pages 115–150.

Goo Google Statistics, IPv6 adoption. URL http://www.google.com/ipv6/statistics.html.

GSK+11 Ghodsi, A., Shenker, S., Koponen, T., Singla, A., Raghavan, B. and Wilcox, J., Information-centric networking: seeing the forest for the trees. *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*. ACM, 2011, page 1.

Han06 Handley, M., Why the internet only just works. *BT Technology Journal*, 24,3(2006), pages 119–129. URL http://dx.doi.org/10.1007/s10550-006-0084-z.

HTSC08 Hunkeler, U., Truong, H. L. and Stanford-Clark, A., MQTT-S – A publish/subscribe protocol for Wireless Sensor Networks. *Communication Systems Software and Middleware and Workshops, 2008. COMSWARE 2008. 3rd International Conference on*, 2008, pages 791–798.

ICA11 ICANN, Available pool of unallocated ipv4 internet addresses now completely emptied, February 2011. URL `http://www.icann.org/en/news/releases/release-03feb11-en.pdf`.

Jac06 Jacobson, V., A new way to look at networking, Google Tech Talks, August 2006.

JST+09 Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H. and Braynard, R. L., Networking named content. *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, CoNEXT '09, New York, NY, USA, 2009, ACM, pages 1–12, URL `http://doi.acm.org/10.1145/1658939.1658941`.

KCC+07 Koponen, T., Chawla, M., Chun, B.-G., Ermolinskiy, A., Kim, K. H., Shenker, S. and Stoica, I., A data-oriented (and beyond) network architecture. *Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '07, New York, NY, USA, 2007, ACM, pages 181–192, URL `http://doi.acm.org/10.1145/1282380.1282402`.

KRS+09 Kumar, P., Reinitz, H., Simunovic, J., Sandeep, K. and Franzon, P., Overview of rfid technology and its applications in the food industry. *Journal of Food Science*, 74,8(2009), pages R101–R106. URL `http://dx.doi.org/10.1111/j.1750-3841.2009.01323.x`.

LDB03 Leopold, M., Dydensborg, M. B. and Bonnet, P., Bluetooth and sensor networks: a reality check. *Proceedings of the 1st international conference on Embedded networked sensor systems*, SenSys '03, New York, NY, USA, 2003, ACM, pages 103–113, URL `http://doi.acm.org/10.1145/958491.958504`.

LS01 Liu, J. and Singh, S., Atcp: Tcp for mobile ad hoc networks. *Selected Areas in Communications, IEEE Journal on*, 19,7(2001), pages 1300–1315.

MM05 Michael, K. and McCathie, L., The pros and cons of rfid in supply chain management. *Mobile Business, 2005. ICMB 2005. International Conference on*, 2005, pages 623–629.

Mul07 Mulligan, G., The 6LoWPAN architecture. *Proceedings of the 4th workshop on Embedded networked sensors*, EmNets '07, New York, NY, USA, 2007, ACM, pages 78–82, URL `http://doi.acm.org/10.1145/1278972.1278992`.

NDN NDN, *Named data networking*. URL `http://named-data.net/`.

Net NetINF, *Network of Information*. URL `http://www.netinf.org/`.

Nor09 Norair, J., Introduction to dash7 technologies. *Dash7 Alliance Low Power RF Technical Overview*.

PPJ11 Pan, J., Paul, S. and Jain, R., A survey of the research on future internet architectures. *Communications Magazine, IEEE*, 49,7(2011), pages 26–36.

PUR PURSUIT, *Publish subscribe internet technology*. URL `http://www.fp7-pursuit.eu`.

Rou06 Roussos, G., Enabling rfid in retail. *Computer*, 39,3(2006), pages 25–30.

SB11 Shelby, Z. and Bormann, C., *6LoWPAN: The wireless embedded Internet*, volume 43. Wiley.com, 2011.

SCC13 Sun, W., Choi, M. and Choi, S., Ieee 802.11 ah: A long range 802.11 wlan at sub 1 ghz. *Journal of ICT Standardization*, 1, pages 83–107.

SCFJ03 Schulzrinne, H., Casner, S., Frederick, R. and Jacobson, V., RTP: A Transport Protocol for Real-Time Applications, RFC 3550 (INTERNET STANDARD), July 2003. URL `http://www.ietf.org/rfc/rfc3550.txt`. Updated by RFCs 5506, 5761, 6051, 6222, 7022.

SCT Stanford-Clark, A. and Truong, H. L., MQTT For Sensor Networks – Protocol Specification, Version 1.2. URL http://mqtt.org/MQTT-S_spec_v1.2.pdf.

SHB13 Shelby, Z., Hartke, K. and Bormann, C., Constrained application protocol (CoAP).

Stu09 Sturek, D., ZigBee IP stack overview. *ZigBee Alliance*.

TBY+12 Tao, H., Brenckle, M. A., Yang, M., Zhang, J., Liu, M., Siebert, S. M., Averitt, R. D., Mannoor, M. S., McAlpine, M. C., Rogers, J. A. et al., Silk-based conformal, adhesive, edible food sensors. *Advanced Materials*, 24,8(2012), pages 1067–1072.

The There Corporation, ThereGate specification. URL http://www.therecorporation.com/en/system/files/media_files/ThereGate_specifications.pdf.

Wyl06 Wyld, D. C., RFID 101: The next big thing for management. *Management Research News*, 29,4(2006), pages 154–173.

XVS+13 Xylomenos, G., Ververidis, C., Siris, V., Fotiou, N., Tsilopoulos, C., Vasilakos, X., Katsaros, K. and Polyzos, G., A survey of information-centric networking research, 2013.

YMG08 Yick, J., Mukherjee, B. and Ghosal, D., Wireless sensor network survey. *Computer Networks*, 52,12(2008), pages 2292 – 2330. URL http://www.sciencedirect.com/science/article/pii/S1389128608001254.