Capillary Networks – Bridging the Cellular and IoT Worlds

Oscar Novo, Nicklas Beijar, Mert Ocak, Jimmy Kjällman, Miika Komu, Tero Kauppinen
Ericsson Research
Jorvas, Finland
Email: firstname.lastname@ericsson.com

***Abstract*—The Internet of Things (IoT) represents a new revolutionary era of computing technology that enables a wide variety of devices to interoperate through the existing Internet infrastructure. The potential of this era is boundless, bringing in new communication opportunities in which ubiquitous devices blend seamlessly with the environment and embrace every aspect of our lives. Capillary networks will be a fundamental part of the IoT development, enabling local wireless sensor networks to connect to and efficiently use the capabilities of cellular networks through gateways. As a result, a vast range of constrained devices equipped with only short-range radio can utilize the cellular network capabilities to gain global connectivity, supported with the security, management and virtualization services of the cellular network. This paper introduces a new Capillary Network Platform and describes the rich set of functionalities that this platform enables. To show their practical value, the functionalities are applied to a set of typical scenarios. The aim of this paper is to give the reader insight about the Capillary Network Platform and illustrate how this work can be used to enhance existing IoT networks and tackle their problems.**

Keywords*—*M2M, Internet of Things, Capillary Networks, Cloud Execution Environment, Self-organizing Networks.

I. Introduction

In recent years, the Internet of Things (IoT) [1] is becoming a fundamental part of the Internet due to its ubiquitous presence, having an accelerating influence in our society. With the advent of IoT, we start to experience the proliferation of new constrained devices around us, which encompass a diverse group of new wireless technologies (e.g., Bluetooth Low Energy [2], IEEE 802.11ah [3], IEEE 802.15.4 [4]). The opportunities resulting from diversified connectivity will enable the creation of new markets. This includes everything from health, automotive, home automation or wearables. It is expected that all these will introduce new requirements on the supporting infrastructure, which will ultimately lead to new feasible IoT platforms to facilitate the integration of IoT into the current Internet.

Cellular communication technologies can play a crucial role in the development and expansion of IoT. Cellular networks can leverage their ubiquity, integrated security, network management and advanced backhaul connectivity capabilities into IoT networks. In this regard, *capillary networks* [5]–[7] aim to provide the capabilities of cellular networks to constrained networks while enabling connectivity between wireless sensor networks and cellular networks. Hence, a capillary network provides local connectivity to devices using short-range radio access technologies while it connects to the backhaul cellular network through a node called Capillary Gateway (CGW). This architecture is shown in Figure 1.

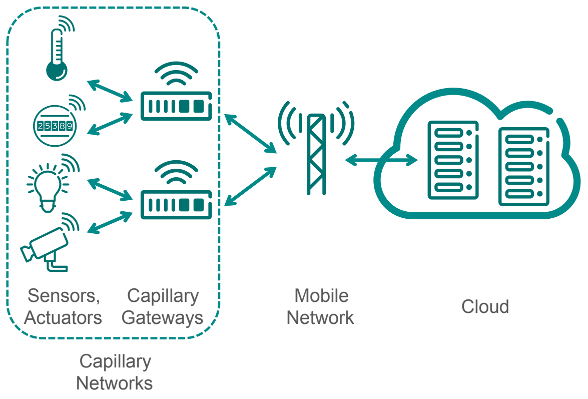

Fig. 1. Capillary Networks

The *Capillary Network Platform (CNP)* differs from today's typical over-the-top solutions in several ways. Our platform does not only cover the gateway and cloud components, but also enriches the functionality with support from the cellular network. An example of such functionality is application-layer multicasting, which allows a vast number of devices to easily be deployed, managed and configured in a scalable way. The platform integrates the security and management of the cellular network, offering an end-to-end solution for security and connectivity. In addition to connectivity, the Capillary Network Platform offers a multi-tenant PaaS (Platform as a Service) solution on top of which multiple users, as well as operators, can build and customize services according to their own specific needs. As our distributed cloud spans all the way out to the gateways, software can be deployed there where bandwidth utilization and application latency need to be optimized.

The key concepts of the Capillary Network Platform were already introduced earlier [7]. However, the earlier work was a conceptual study and it did not include any implementation details. This paper presents a thorough description of the architecture of our Capillary Network Platform, particularly focusing on the main new features offered by the platform. The paper also describes the technical solutions of our implementation, which was presented in the Mobile World Congress 2015[1].

[1]http://www.mobileworldcongress.com/

The paper is structured as follows. Section II analyses the requirements of our platforms. Section III describes the Capillary Network Platform architecture, while Section IV explains the main functionalities of the architecture. Section V presents details of our prototype implementation. Section VI presents an analysis of applying the Capillary Network Platform to typical IoT scenarios. The last section summarizes the paper and outlines the directions for further work.

II. Background

IoT platforms aim to support easy deployment of IoT applications for networks of connected devices. Such platforms support several types of nodes, communication protocols and technologies, where the nodes can range from simple sensor devices with limited capabilities to more powerful nodes, which can act as gateways or aggregators. A comparison of several IoT platforms is given by Mineraud et al [8].

Current IoT platforms have the ability to abstract away implementation components at lower layers. This over-the-top approach might be preferred for layer independence and simplification in many cases. However, several situations remain where a network-aware infrastructure can improve the efficiency and facilitate the interaction and capabilities of IoT services.

Unlike the traditional IoT platforms, the Capillary Network Platform is a network-aware platform, in which intelligence is introduced at different levels to the network resulting in a more flexible approach. Thus, the network is not only a transparent pipe between a device and the data storage, but provides support for higher-level services. The bottom layers provide a gateway selection mechanism for load-balancing and a virtualization mechanism for middleware services, the middle layers provide an orchestration and a computational data engine, and the top layer provides a device management and service-enablement interface.

A. Requirements

Our vision of the Capillary Network Platform has been designed to meet a set of fundamental requirements. The most important ones are the following:

- *Device Mobility*: The network should be able to direct the sensor devices to the optimal capillary gateway in order to achieve a specific goal, such as to minimize the delay, maximize the availability or to provide load balancing, while considering the quality of the radio connection.
- *Scalable Management*: The platform as a whole should be able to address and manage multiple devices in groups in order to reduce the complexity of device management.
- *Cloud-oriented Architecture*: The functionalities of the network should be implemented as virtual instances in the cloud in order to improve flexibility and fast deployment of IoT services.
- *Automated, Fine Granularity Instantiation*: The platform should provide the possibility for automatic deployment of IoT service instances in the cloud for specific devices or gateways.
- *Nearby Instantiation*: While data centers are connected with high-bandwidth links, their distant location from the devices causes inherent latency. Running some processing close to the device lowers the delay and delay variation in the control loop, and minimizes the amount of network transmission. Therefore, IoT services should, in given cases, be placed as close as possible to the device.
- *Automatic Configuration*: Typically, capillary network deployments do not involve any network planning. Capillary networks should be able to configure themselves and should be easy to deploy.
- *Security*: Highly constrained devices may need communication security in some scenarios. Besides communication security, the system should also consider other related security issues, including authentication and identification of gateways and devices.

III. Architecture

The capillary network connects constrained devices via multiple short-range radio technologies, such as Bluetooth Low Energy [2], IEEE 802.15.4 [4] or IEEE 802.11ah [3], to Capillary Gateways (CGW). CGWs provide connectivity via a backhaul network (typically a 3GPP cellular network) to the cloud, where data storage, processing and network management is implemented. CGWs may be operated by an operator or by a third party.

In order to enable wireless sensor networks to connect and use the capabilities of cellular networks efficiently, we have developed and implemented a high-level network architecture, which is depicted in Figure 2. The architecture has been designed for a cloud environment, so the boxes in the figure indicate software components rather than physical elements. The main target is to integrate the Capillary Network Platform into a cloud-enabled operator network, thus, the operator domain covers all the cloud components. The architecture supports multiple tenants, which can run customized functions, called IoT services, for data processing and to control of their devices.

On a high level, the network is divided into logical entities with the following responsibilities:

- *Devices* are constrained devices connected via a short-range radio technology, such as IEEE 802.15.4 or Bluetooth Low Energy, as mentioned above. The devices expose a set of IPSO objects [9] for accessing its sensors and actuators. Depending on the device type, the devices provide a CoAP/LWM2M [10] interface, or a HTTP interface for management, actuation and data access.
- *Capillary Gateways (CGW)* provide connectivity between the short-range network and the cellular network. The CGW contains software components for registration of the CGW and the devices, a Local Connectivity Manager for locally managing the connectivity between devices and CGWs, and a proxy

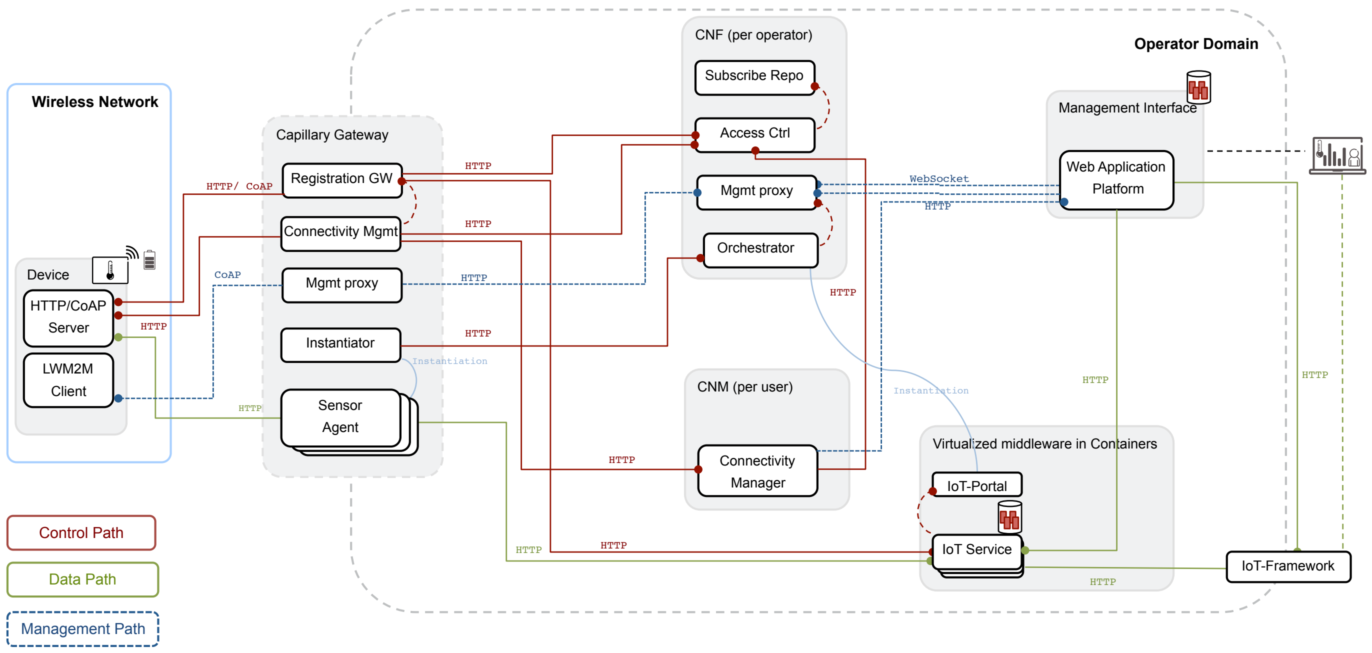

Fig. 2. Logical Architecture of the Capillary Network Platform

for relaying and distributing management commands. Furthermore, CGWs that participate in a distributed cloud can run Sensor Agents, which are IoT service instances specific to physical sensors. These sensor agents are controlled by an Instantiator on the CGW.

- *Capillary Network Function (CNF)* is the operator's control function for capillary networks. It controls the registration and authentication of both CGWs and devices. It interfaces with the Operations & Support System (OSS) for the 3GPP network and, in particular, enforces management rules and policies based on subscription data. CNF also orchestrates the cloud instances and instantiates software images based on the subscriber's configuration.
- *Capillary Network Management (CNM)* includes tools and services to support automatic configuration and management of the capillary networks of a particular tenant. In particular, it contains the Connectivity Management Server, which automates the decisions related to connectivity between devices and CGWs, supporting mobility of devices between all the tenant's CGWs.
- *Management Interface (MINT)* is the network-side functionality that performs device management for the devices and CGWs by interfacing with the network manager. Device management includes configuration, status reporting and actuation among other functionalities.
- *Virtualized Middleware in Containers* is the virtualization platform for IoT Services, which provides data processing and actuator control functions in the cloud. These IoT Services are instantiated and managed by the IoT-Portal. A particular type of IoT Service, a "Virtual Capillary Gateway", can be automatically created by the CNF for each physical CGW to provide middleware functions in the cloud. One of the advantages of such Virtual Capillary Gateways is that they can provide an interface to the CGW, including access to buffered data, even when a CGW is not currently reachable.
- *IoT-Framework* [11] is a data store and a computational engine for quantitative information accumulated by connected devices [12]. The IoT-Framework can be provided by the operator, or alternatively by an operator independent third party.

IV. Functionalities

This section identifies the major functionalities and explains their implementation using the above logical entities.

A. Bootstrapping and Security

Before a CGW can connect to the platform and start communicating with an IoT Service, it must authenticate itself to the CNF, where *Access Controller* function grants access to the service. One particular authentication mechanism implemented in this platform is based on the *Generic Bootstrapping Architecture* (GBA) by 3GPP [13]. This method can be used by CGWs (or devices) that have 3GPP credentials, i.e., a physical SIM card or software-based SIM and a corresponding mobile subscription.

Figure 3 depicts, on a high level, the process by which CGWs are authenticated and registered to use cloud services. Initially, a CGW connects to the GBA *Network Application Function* (NAF), which is implemented by the CNF in our platform. The NAF instructs the CGW to connect to a GBA *Bootstrapping Server Function* (BSF), and the CGW[2] performs

[2]The CGW has the role of a 3GPP User Equipment (UE) in this procedure.

a secure bootstrapping procedure based on GBA, which includes authentication and generation of shared keys [13], [14]. The keys can be utilized for establishing a secure communication channel between the CGW and CNF. Also communication between the CGW and other cloud functions, e.g., the CNM or the Virtualized Middleware in Containers can use this channel.

After this procedure, the CGW is able to register itself to the *IoT-Portal* in the cloud, and obtains access to an existing or newly-instantiated *IoT Service*. Finally, the CGW registers itself to the IoT Service. Devices, in turn, register themselves to the CGW, which forwards or translates the registration to the IoT Service. The registration of the CGW and device is also sent to the CNM, either directly from the CGW or via the CNF.

Once the registrations are performed, devices can send data to IoT Services. In this context, source authentication and integrity protection of messages carrying data (or registration information) sent from devices can be implemented by using sufficiently lightweight but secure public key based signatures (e.g., ECDSA), as described in more detail by others [14], [15]. On the other hand, we assume that the CGW, the CNF and the CNM belong to the operator's domain and data encryption is not needed between them.

B. *Connectivity Management*

The connectivity management function implemented as part of the CNM allows the connectivity of the devices to be centrally managed. The goals for connectivity management is defined using a policy, which may, for example, specify that load balancing is the main objective for this particular network. The CNM maintains a registry of all connected CGWs and devices. It monitors the properties (e.g. the load level) of the CGWs and the devices and, in particular, keeps track of which CGWs are within the radio reach of each device, and to which CGW the device currently is connected.

Based on the reported radio connectivity, the policy and the CGW's properties, the CNM calculates the optimal distribution of devices between the available CGWs. In allocating devices to CGWs, the properties of the CGWs are considered. For example, CGWs with high bandwidth uplink can be preferred, while battery-powered gateways with low remaining battery power can be avoided. The network administrator controls the connectivity management indirectly by defining a policy, which specifies the relative weights of various CGW's properties to be considered. The optimal distribution of devices is calculated by iterating through all the devices to find a target gateway for each device. To find the best gateway for a device, a preference value $p(g)$ is calculated for each gateway g that is reachable from the device, and the gateway with the highest preference is chosen for that device. The preference is based on the weighted sum of the properties,

$$p(g) = \sum_{c \in C} w_c v_c(g), \tag{1}$$

where w_c is the weight of the property c as defined in the policy, and $v_c(g)$ is the value of the property c of gateway g. The algorithm is described thoroughly by Beijar et al [16]. If a new preferred gateway is found, the CNM automatically sends

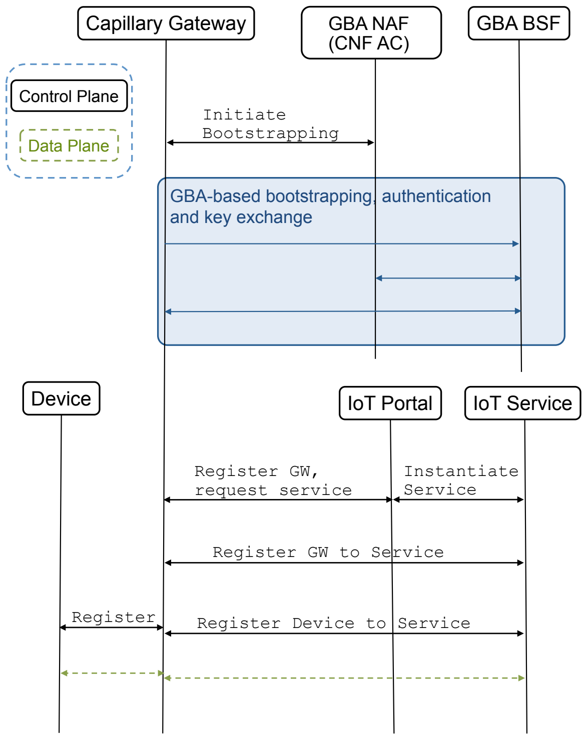

Fig. 3. Bootstrapping, Authentication, and Registration.

a management command to instruct the device to connect to the preferred CGW.

Alternative algorithms can be implemented as plugin modules to the CNM. The CNM can, for example, estimate the movement of a mobile device and provide a list of alternative CGWs along the movement path in the order of preference, which are used when the current CGW becomes unavailable.

The CNM is implemented as a software component that is instantiated separately for each tenant. This allows tenants to customize the management software with new functions and gateway selection algorithms. The CNM interfaces with the CNF for CGW registration and authentication, and with MINT for retrieving policies.

Connectivity management is supported by a Local Connectivity Manager process in the CGW, which unloads the CNM from routine tasks performed locally in the capillary network, e.g., polling the reachability of devices. While some technologies (e.g. IEEE 802.11) provide information about reachability from the device perspective, other technologies (e.g. IEEE 802.15.4 [4] with RPL routing) provide reachability information from the CGW's side. As our CGW support multiple technologies, it merges reachability information from several interfaces. When the CGW detects changes in the radio connectivity information (e.g. a new gateway within the range of the device) or in the CGW properties (e.g. load level changed), it sends an update to the CNM, which calculates a new distribution and, if required, sends management commands to enforce a device to switch to another CGW. The management command is sent via the CGW to which the device currently is connected and it contains the list of target CGWs in the form of IPSO objects [9]. Once a device switches to another CGW, it will register again to the new CGW.

C. Device and Gateway Management

The vast amount of devices and CGWs and their distribution throughout the network create challenges in device management. The network manager requires an overview of all devices and CGWs, and an intuitive interface to control them. Device monitoring and control, capillary network monitoring and management, device or resource identification and discovery are some of the challenges for the Capillary Networks architecture. To solve such challenges, a new network node called the Management Interface (MINT) was introduced into the architecture. MINT is a wrapper layer on top of all the CNP components, i.e., it is the interface between all the network nodes and the network manager as shown in Figure 2. Hence, MINT incorporates different modules in order to provide a complete view of the system.

The core of MINT is a web application server that coordinates the communication between different entities of the Capillary Network Platform (CNF, CNM, IoT-Framework and virtualized middleware) and pub/sub servers, additionally provides a user interface to the network manager. In order to retrieve the management information and device data, as well as to present it to the manager in a map-based interface as shown in Figure 4, MINT communicates with the CNF, CNM, IoT-Framework and virtualized middleware through a large set of RESTful APIs:

- *CNF to MINT:* CNF provides the network topology and management APIs to MINT, while CNF also dispatches device management information from MINT to the CGW.
- *CNM to MINT:* MINT controls CNM features used in Capillary Network Management such as the policy in the connectivity management and the gateway property updates of the CGW. MINT also provides configuration tools for CNM.
- *Virtualized middleware in Containers to MINT:* MINT retrieves the meta data of CGWs and devices from virtualized middleware instead of contacting the device directly. This approach ensures data availability even if the device is unreachable.
- *IoT-Framework to MINT:* IoT-Framework provides the data collected from the devices to MINT. Semantic search through device data is also provided by the IoT-Framework.

RESTful APIs provide stateless response when the manager requests the relevant data. However, a complete management node is required to support real-time network information to the manager. For this requirement, MINT includes a publish/subscribe server. This publish/subscribe server notifies the manager in real-time when an error is encountered in the devices or CGWs, the network topology changes or a new device joins the network. It additionally provides real-time logs from CGWs and data streams from devices through the IoT-Framework.

The direct management or actuation of constrained devices is performed by using OMA DM Lightweight Machine-to-Machine Device Management protocol (LWM2M) [10]. The manager can send LWM2M commands to the device directly,

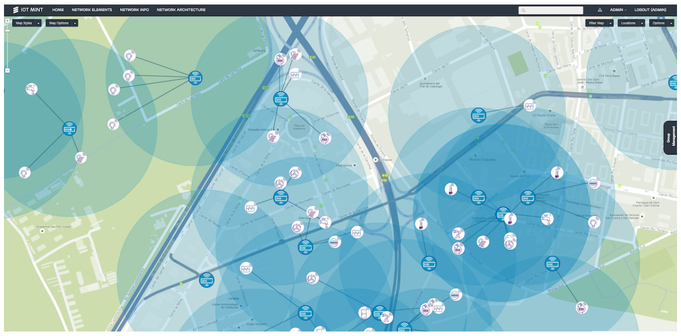

Fig. 4. MINT User Interface

for example, to perform an actuation on the device. Additionally, we extended LWM2M protocol to support scalable management, i.e., application level group communication. In order to provide group communication, a specific Group Object is introduced to LWM2M protocol. Thus, the manager can send a group message (e.g. turning on street lights of a specific area) implemented as a unicast message by MINT to the management proxy in CNF. Once the proxy receives the message, it sends unicast messages to each LWM2M client in the group using the list of group members stored in it. LWM2M application level group communication [17] offers advantages such as increased reliability and does not have special requirement from the network equipment when compared to IPv4/IPv6 multicast.

D. Distributed Cloud

In our system, the distributed cloud for IoT devices includes both data centers (DCs), where larger amounts of data from different sources can be processed and where also management functions typically reside, as well as local compute infrastructure, in particular within capillary networks. The latter makes it possible to process data locally, for example, to aggregate or filter sensor data, which can reduce the amount of data that is sent upstream towards data centers. Moreover, it enables also low-latency sensor-actuator control loops.

In particular, we use *containers*, such as Docker [18], for packaging, deployment (including updates and new features), and execution of software in our cloud. In general, containers have a low overhead, which, in turn, allows a high density of instances in DCs. Moreover, containers can be executed in more constrained environments without hardware virtualization support, such as in local CGWs.

The *Orchestrator* component of the CNF is the high-level orchestration service, determining what software image is deployed for a CGW or device when it registers. It also determines whether the software should be deployed locally on a CGW or in the Virtualized middleware.

The *Virtualized middleware* domain is managed by the *IoT-Portal*. The IoT-Portal is a Docker orchestration service that instantiates, configures and manages instances in the virtualized middleware, i.e., IoT Services. We typically allocate one IoT Service instance per CGW for initial data processing.

Nowadays, the Docker ecosystem provides a wide range of orchestration solutions [19]. The IoT-Portal was created due to a lack of mature IPv6 support in both Docker and in the related orchestration tools at the time of implementing the Capillary Network Platform. In order to support a potentially vast number of devices, the IoT Service containers need to have IPv6 connectivity in practice. IPv6 support in Docker has earlier been immature, which has led us to enhance the network functionality in the prototype's Docker environment. The IoT-Portal uses the IPv6 prefix of an underlying Docker host and IPv6 Neighbor Discovery Proxy [20] functionality to assign the container a unique IPv6 address that is routed to the container without requiring the network administrator to allocate an IPv6 prefix to each Docker host.[3]

Some of the CGWs support virtualization and can run containers with data processing software near the device. We call these instances Sensor Agents, since we typically allocate one virtual counterpart per sensor device. The Orchestrator communicates with an Instantiator on the CGW in order to launch a new container. If the CGW to which the device is connected does not support virtualization, the Sensor Agent can be started in the data center instead.

The platform also supports launching of emulated devices inside Docker containers in the virtualized middleware for diagnosing and stress-testing the entire system. The emulated sensors communicate using the same protocols as physical sensor devices.

V. Implementation

We developed a prototype of the Capillary Network Platform in order to test and evaluate the concept. The following section provides additional details about our implementation, in particular regarding the gateways, the devices and the components in the cloud.

A. Gateways

Gateways are Linux based platforms running customized software to add the features of a Capillary Gateway. The short range radios and the mobile uplink are mainly provided with USB dongles. We used two types of gateways:

- *TP-Link TL-WR1043ND* gateways are modified to run the Linux based OpenWRT[4] operating system. The integrated access point serves an IEEE 802.11 based short range network. The single USB port is used either for the 3G modem or for serving an IEEE 802.15.4 short range network via a dongle[5]. The OpenWRT platform does not currently support Docker based virtualization.
- *Raspberry Pi 1 Model B* gateways run the Arch Linux[6] operating system. Both IEEE 802.11 and 802.15.4 short-range radio interfaces, as well as the 3G modem, are based on USB dongles. Arch Linux supports Docker for the ARM platform, which enables virtualization at the gateway.

[3]As an alternative solution, the IPVLAN driver [21], which is included in recent Linux kernels, could potentially be utilized for sharing a Layer 2 network device on the host with containers.

[4]http://openwrt.org/

[5]A USB hub can be used to enable both simultaneously

[6]http://www.archlinux.org/

B. Devices

To show support of various kinds of devices, our implementation utilizes two very different types of devices:

- *TMote Sky*[7] devices with the Contiki[8] operating system running 6LoWPAN [22] over an IEEE 802.15.4 interface with CoAP at the application layer. We base our device implementation on the RPL (IPv6 Routing Protocol for Low-Power and Lossy Networks) [23] routing protocol between the gateway and the device. RPL implements the messaging to allow the capillary gateway and the device to detect each other. The reachability information is obtained from the gateway's perspective. The TMote Sky devices integrate humidity, temperature, and light sensors but only the temperature and humidity sensors were used in our implementation.
- *Libelium Waspmote*[9] devices are connected with IEEE 802.11b/g [3] and use HTTP for communication. The devices perform a scan over all channels to find the SSIDs of the surrounding gateways. Thus, they provide a device centric view of the reachability. The Waspmote devices are equipped both with sensors and actuators in the prototype. Some of those actuators are motors and lights. The actuators are connected to the Waspmote devices through the general-purpose input/output (GPIO) interface by controlling the voltage from 0V to 3.3V. External sensors are connected to the analogue input ports.

C. Cloud components

The cloud components are implemented as software running in Docker containers (CNF, CNM, IoT-Portal, IoT-Framework) or in a virtual machine (MINT). These components implement the functions described in section IV.

- *CNF* is implemented as a web service running on the Apache server[10] with MySQL[11] data storage.
- *CNM* is implemented on the Twisted[12] framework with a Redis[13] based data storage to save information about gateways, devices, policies and reachability. The CNM provides APIs as plugin modules for various kinds of devices. The core of the CNM is the connectivity management function that implements the gateway selection algorithm.
- *IoT-Portal* is implemented using the Apache server with MySQL as data storage.
- *MINT* is implemented as a web service consisting of two separate servers, which are an Apache server combined with MySQL storage and a Node.js[14] server as a publish/subscribe service for performing real-time

[7]http://www.moteiv.com/

[8]http://contiki-os.org/

[9]http://www.libelium.com/

[10]http://httpd.apache.org/

[11]http://www.mysql.com/

[12]http://twistedmatrix.com/

[13]http://redis.io/

[14]http://nodejs.org/

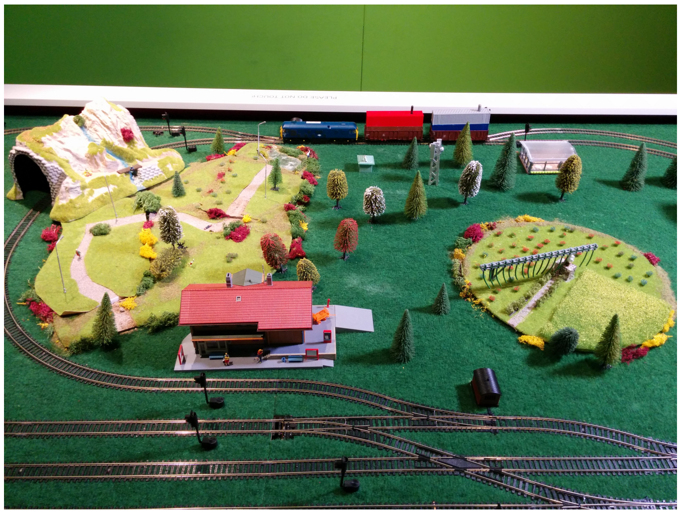

Fig. 5. The use cases implemented with the prototype

updates on the client. Data transfer between these two servers are handled by local Redis data queues.

- *IoT-Framework* is a database implemented in Erlang by utilizing the Elasticsearch engine[15]. The framework has been implemented in collaboration with the Swedish Institute of Computer Science and Uppsala University and it is available as open source software in GitHub [12], released under the Apache 2.0 license.

VI. Use Case based Evaluation

The Capillary Network Platform is focused on enabling a wide range of cloud services for IoT devices as well as an easy deployment and management of network elements. Our goal with the prototype was to evaluate the different functionalities of the platform by creating real-world use cases and testing them empirically. This section describes how the platform supports these use cases. The use cases are tested using a miniature model setup shown in Figure 5, where sensor devices are placed in a train's containers and in an agriculture field, while streetlights, greenhouse lights and an irrigation system are controlled by actuator devices.

For the case of distributed cloud, a typical use case we considered is a control process reading and analysing the data from a sensor and, based on the analysis, performs some actuation with the device. While data centers are connected with high bandwidth links, their distant location from the device can cause inherent latency. Running the processing close to the device lowers the delay and the delay variation in the control loop. In addition, placing the processing at the CGW also improves reliability and gives a degree of autonomy; if the uplink from the CGW goes down, the control process can continue operating. Moreover, sensor data typically has a high degree of redundancy. With filtering, compression and aggregation performed on the CGW, the amount of data transmitted over the wireless uplink can be reduced. While this functionality could be performed with traditional custom-built control software statically installed on the CGW, running the software as part of a distributed cloud provides flexibility, easier management and better interactions with other systems. In the prototype, the distributed cloud is demonstrated by running the software processing sensor data in an agriculture use case, where actuation is performed with a low latency when the software determines that an action is needed based on sensor input.

[15]http://www.elastic.co/products/elasticsearch

The distributed cloud can further be used to offload a part of the sensor software to the cloud. The sensor software is then split into two parts: the firmware of the physical device and the corresponding driver in the cloud, which may run, e.g., on the CGW. When the device joins the network, the device type is identified and the right driver is automatically instantiated in the cloud. This saves memory and processing of the device, lowering its price, and making it more generic.

In the group management use case, semantic information about the devices is used to sort the different devices in groups. Those groups could be defined based on any of the attributes a device has, e.g., sensor type or manufacturer name. When a manager creates a group, a logical relation is created between a set of devices, which causes them to subscribe to the messages for those groups. The information about the group membership is stored in management proxies on several layers in the network, e.g. in the cloud and in the CGW. This way, when a message is sent to the group, it is sent via the proxies in a efficient manner, with low overhead, it is aggregated and prioritized according to importance. In the prototype, multiple streetlights form a group that can be controlled and managed without addressing individual devices. For example, a single command can be sent to switch on all streetlights in the group.

The connectivity management use case [16] takes multiple factors, such as the CGW load and battery level, into account in order to select the optimal gateway for each device. If the CGW load were the only property, the implemented algorithm would choose the gateway with less load as the best candidate. When several properties are available, the algorithm considers all properties according to the weight specified in the policy by the network manager. Our prototype considers the weight and battery properties, but a real world implementation could consider any property that can be given a numerical value, such as the uplink bandwidth or reliability. Two types of devices are evaluated in our implementation: IEEE 802.15.4 based devices with RPL routing and IEEE 802.11b/g based devices. The former obtain the reachability information from the CGW sending the list of reachable devices to the CNM, while the latter obtain that information from the devices sending the list of the reachable CGWs to the CNM. In the prototype, this scenario is demonstrated with sensors located on a train, which can move between the radio range of multiple CGWs. The platform connects the sensors to the optimal gateway among the reachable ones based on the load and battery properties of the CGW and the weight defined in the policy.

VII. Conclusions and Future Work

This paper introduces the Capillary Network Platform (CNP) for bridging the Internet of Things and the cellular network. The platform provides backhaul cellular connectivity to constrained networks and, unlike the traditional IoT platforms, which abstract away the lower layer, adds intelligence to the network at multiple levels resulting in a more flexible and cross-layer approach. In particular, integrating connectivity management, device management and security solutions of both network domains enables an end-to-end solution with a similar automatic configuration and easy deployment offered to the IoT devices as we are used to in cellular devices today. In addition, the Capillary Network Platform, by leveraging distributed cloud technology, provides multiple users and operators the opportunity to deploy their own software for data processing and to customize the service to their own specific needs.

Our goal was to provide a generic, scalable, and easily manageable platform for IoT services. Hence, our implementation is heavily based on enabling distributed cloud services, group management and load balanced connectivity, as presented in this paper. We argue that the Capillary Network Platform is suitable for various typical IoT scenarios. The current prototype implementation has served us well in multiple IoT scenarios, and has been publicly demonstrated in the Mobile World Congress 2015 to a large number of participants.

However, parts of this proposed architecture are still subject for ongoing research. We are currently improving orchestration, developing redundancy as well as integrating distributed IoT data processing solutions, such as Calvin [24], for aggregating, filtering and preprocessing data at the edge. During our work, standardization activities have also been ongoing, creating a need to flexibly to adopt new interfaces and protocols, a task that can be met by the use of virtualization and abstraction in the platform.

ACKNOWLEDGEMENT

The authors would like to thank Jari Arkko, Jan Melén, Jaime Jiménez and Petri Jokela for their feedback on this paper. This work was partly carried in the IoT SRA program of Tivit (Finnish Strategic Centre for Science, Technology and Innovation in the field of ICT), funded by Tekes.

REFERENCES

[1] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A Vision, Architectural Elements, and Future Directions," *Future Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, Sep. 2013.

[2] "Bluetooth standards." [Online]. Available: https://www.bluetooth.org/en-us/specification/adopted-specifications

[3] "IEEE 802.11 standard." [Online]. Available: http://www.ieee802.org/11/

[4] "IEEE 802.15 standard." [Online]. Available: http://www.ieee802.org/15

[5] S. Singh and K.-L. Huang, "A Robust M2M Gateway for Effective Integration of Capillary and 3GPP Networks," in *5th International Conference on Advanced Networks and Telecommunication Systems (ANTS)*. IEEE, 2011, pp. 1–3.

[6] V. Misic, J. Misic, X. Lin, and D. Nerandzic, "Capillary Machine-to-Machine Communications: The Road Ahead," in *Ad-hoc, Mobile, and Wireless Networks*, 2012, vol. 7363, pp. 413–423.

[7] J. Sachs, N. Beijar, P. Elmdahl, J. Melen, F. Militano, and P. Salmela, "Capillary network - a smart way to get things connected," in *Ericsson Review*, 2014. [Online]. Available: http://www.ericsson.com/res/thecompany/docs/publications/ericsson_review/2014/er-capillary-networks.pdf

[8] J. Mineraud, O. Mazhelis, X. Su, and S. Tarkoma, "Connecting Internet-of-Things Deployments Across Space and Time," in *IEEE IoT*, 2013.

[9] "IP for Smart Objects Alliance." [Online]. Available: http://www.ipso-alliance.org/

[10] *Lightweight Machine to Machine, Technical Specification*, Open Mobile Alliance (OMA), 2014.

[11] K. Vandikas and V. Tsiatsis, "Performance Evaluation of an IoT Platform," in *Eighth International Conference on Next Generation Mobile Apps, Services and Technologies (NGMAST)*, 2014.

[12] "IoT Framework." [Online]. Available: http://github.com/EricssonResearch/iot-framework-engine

[13] *TS 33.220 - Generic Bootstrapping Architecture (GBA)*, 3GPP. [Online]. Available: http://www.3gpp.org/DynaReport/33220.htm

[14] H. Mahkonen, T. Rinta-aho, T. Kauppinen, M. Sethi, J. Kjällman, P. Salmela, and T. Jokikyyny, "Secure M2M Cloud Testbed," in *MOBICOM'13*, 2013, pp. 135–138.

[15] M. Sethi, J. Arkko, and A. Keränen, "End-to-end Security for Sleepy Smart Object Networks," in *LCN Workshops 2012*. IEEE, 2012, pp. 964–972.

[16] N. Beijar, O. Novo, J. Jiménez, and J. Melen, "Gateway Selection in Capillary Networks," in *The 5th International Conference on the Internet of Things (IoT 2015)*, 2015, accepted for publication.

[17] D. D'Ambrosio, "A Group Communication Service for OMA Lightweight M2M," Master's thesis, Universit degli Studi di Napoli Federico II, Italy, 2014.

[18] "Docker Open Platform for Distributed Applications." [Online]. Available: http://www.docker.com

[19] K. Subramanian, "Docker Ecosystem," Mind Map, 2015. [Online]. Available: https://www.mindmeister.com/389671722/docker-ecosystem

[20] D. Thaler, M. Talwar, and C. Patel, "Neighbor Discovery Proxies (ND Proxy)," RFC 4389 (Experimental), Internet Engineering Task Force, Apr. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4389.txt

[21] M. Bandewar, *IPVLAN Driver HOWTO*, Linux Kernel Documentation, 2014. [Online]. Available: https://www.kernel.org/doc/Documentation/networking/ipvlan.txt

[22] J. Hui and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks," RFC 6282 (Proposed Standard), Internet Engineering Task Force, Sep. 2011. [Online]. Available: http://www.ietf.org/rfc/rfc6282.txt

[23] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J. Vasseur, and R. Alexander, "RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks," RFC 6550 (Proposed Standard), Internet Engineering Task Force, Mar. 2012. [Online]. Available: http://www.ietf.org/rfc/rfc6550.txt

[24] P. Persson and O. Angelsmark, "Calvin Merging Cloud and IoT," in *6th International Conference on Ambient Systems, Networks and Technologies (ANT)*, 2015.