Gateway Selection in Capillary Networks

Nicklas Beijar, Oscar Novo, Jaime Jiménez and Jan Melen
Ericsson Research
Jorvas, Finland
Email: firstname.lastname@ericsson.com

***Abstract*—The world is adopting machine-type communication, wherein sensors and actuators blend seamlessly with the environment around us, bringing a new ubiquitous computing and communication era – a shift that is, to some extent, illustrated by the explosive growth of the Internet of Things (IoT). Capillary Networks play an important role in the growth of IoT, enabling wireless sensor networks to connect and use the capabilities of cellular networks through Capillary Gateways. In that sense, Capillary Gateways facilitate the seamless integration of wireless sensor networks with cellular networks. Therefore, an optimal selection of the Capillary Gateways by the wireless sensor network is crucial for balancing the load between the gateways and optimizing the end-to-end path through both networks. This paper describes a set of possible gateway selection architectures and presents an algorithm for determining the gateway selection based on policies and a set of constraints. Then, the paper describes our implementation of two selected architectures, discussing the solutions and challenges encountered during implementation. Finally, the paper evaluates the traffic and load generated by gateway selection.**

***Keywords*—M2M, IoT, Internet of Things, Capillary Network, self-organizing networks.**

I. Introduction

With the growing presence of short-range radio-access technologies and the proliferation of multiple heterogeneous devices connected to Internet, the society is moving towards a new vision, where a wide range of objects become part of the Internet. The decreasing connectivity and technology costs, as well as the increasing network penetration, are key factors enabling that vision called the Internet of Things [1]. Internet of Things (IoT) encompasses any device that utilizes embedded technology to communicate and interact with other devices via Internet. This includes everything from automotives, environmental sensors, and industrial machinery to home appliances and wearable devices. The application possibilities are endless and will profoundly change existing industries such as health, logistics, agriculture and manufacturing as well as create new unforeseen ones. Although IoT is emerging as a technology, the current communication paradigm will need to go beyond the traditional communication scenarios – which have primarily been built to provide connectivity for human interaction – and evolve into a scenario of billions of ubiquitous interconnected devices.

According to market predictions [2], the majority of the IoT devices are expected to use short-range radio technologies such as Bluetooth Low Energy [3], IEEE 802.15.4 [4] or IEEE 802.11ah [5]. In this case, cellular networks can play a valuable role in connecting the short-range radio networks, leveraging on its ubiquity and advanced connectivity for backhaul, and integrating security and network management into the scene.

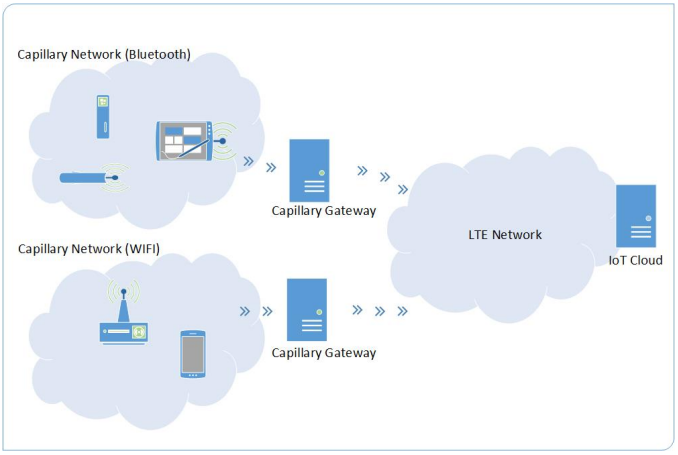

Fig. 1: Capillary networks

The short-range network would act as the fine-grained last-hop network connecting the tiniest devices. Thereby, they are referred to as Capillary Network [6], [7], comparing them with the tiny blood vessels connecting cells to the arteries. Capillary Networks go beyond sensor networks, with not only sensor data being transferred from devices to the cloud, but also cloud applications being able to perform actuation on the devices.

Capillary networks consist of a set of devices connected using short-range radio access technologies to a more powerful device called Capillary Gateway. The gateway connects the devices to the cellular backhaul network, transporting data to an IoT Cloud service, which aggregates the incoming data and manages devices and gateways. Figure 1 illustrates a communication model between different capillary networks (WIFI, Bluetooth) and two capillary gateways using the same cellular network as backhaul (LTE). The sensor data is sent to an IoT Cloud service that offers data aggregation and storage. The IoT Cloud service also enables operators to offer services to enterprise customers such as connectivity management of the devices.

Typically, Capillary Network deployments do not involve any network planning. The networks should be able to automatically configure themselves in order to be easy to deploy. This typically implies over-provisioning the network with redundant gateways, which may have different properties in terms of connectivity, capacity and – in the case of battery powered gateways – also energy. Given the assumption that there are multiple capillary gateways within reach offering communication services with different properties, the capillary network should be able to direct the sensors to the best capillary gateway in order to achieve a specific goal. Examples of

such goals can be minimizing latency, maximizing availability or providing load balancing. The mechanism that leads sensors to the best capillary gateway is called *Gateway Selection*. This mechanism reduces manual configuration of network connectivity, provides redundancy of gateways and optimizes the communication path across network technology borders.

This paper presents and evaluates various solutions for gateway selection. Specifically, Section II presents three different architectures based on the required information, while Section III further details the mechanism for calculating the selection. Section IV applies the solution to existing protocols and describes our prototype implementations. Different approaches taken in the implementation of the gateway selection mechanism are evaluated in section V, and Section VI concludes the paper.

II. Gateway Selection Architecture

The gateway selection process depends on various types of information that must be collected from the capillary network. These *gateway selection parameters* are the input for the gateway selection algorithm. Several *architectures* are possible, depending on where in the network the selection process is executed. The choice of location affects the need to transport information and thereby the performance. This section discusses the gateway selection parameters along with the location of the gateway selection process.

A. Gateway Selection Parameters

Gateway selection relies on three types of information: reachability information, constraints and policy. According to these three parameters, the gateway selection algorithm selects the best candidate for the capillary network.

1) Reachability information: Reachability information describes the possible connectivity between the IoT devices in a capillary network and the capillary gateways around them. The possibility of a link is determined by the existence of a radio signal with sufficient strength in the short-range network. The reachability information is therefore, at it simplest, described as a set of possible connectivity links. However, the information can be augmented with link quality data such as signal strength, uptime or packet loss for each link. The reachability information can be detected by either the gateway or by the IoT devices. IoT devices typically detect connectivity by listening to the beacon signals that gateways transmit.

2) Constraints: The constraint information describes the properties of the network and the nodes that are included in the selection process. Gateway properties can include the load of the gateway (described as the processor load or by the number of connected devices), the remaining battery level, or the cost of use (backhaul costs, transit fees). Constraints describing the cellular link quality and bandwidth allow selecting a gateway with good uplink, so that the path through both network types can be optimized. Most constraints can be specified on a per gateway basis.

3) Policy: The policy determines the goal of the gateway selection. Fundamentally, a policy is a set of priorities that determines how the various constraints affect the best choice of gateways. Policies are static and are defined by the network management. The policy model is described in more detail in III-A.

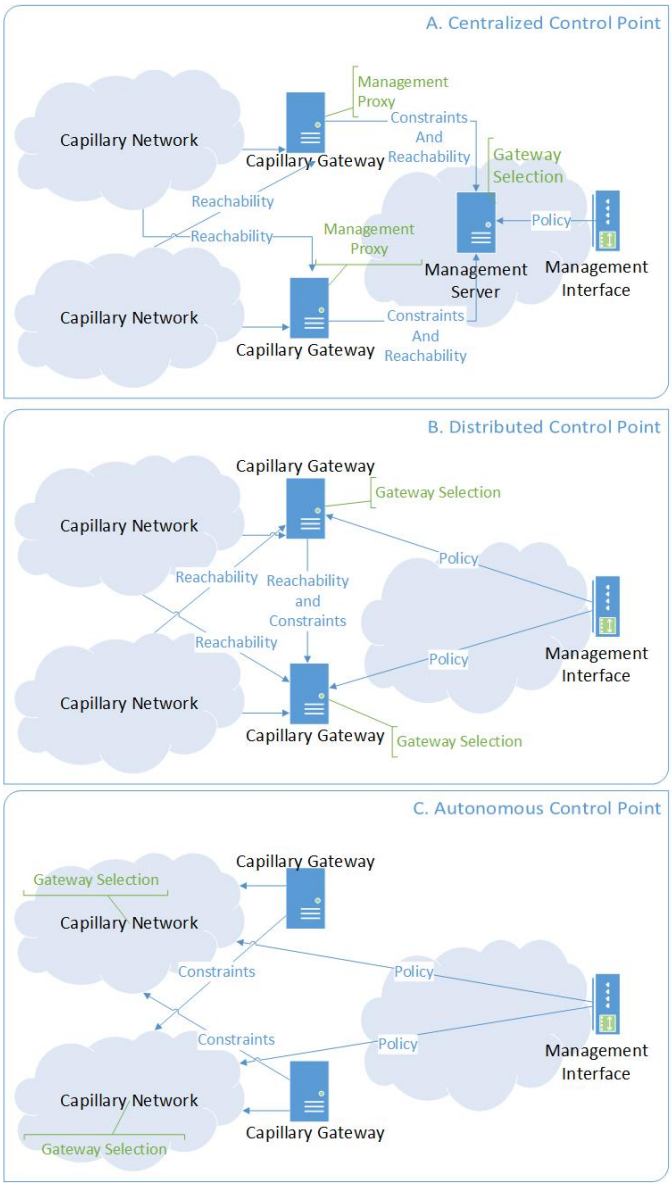

Fig. 2: Control Point Architecture

B. Location of the Gateway Selection Process

The gateway selection process can be controlled at various locations in the network. The control point can be centralized, distributed between the gateways, or distributed between the devices. The location of the node(s) that controls the selection process affects the need to transport the gateway selection parameters to different places in the network. Below, the paper discusses the alternatives in more detail:

1) Centralized: In the *centralized* alternative, there is a centralized gateway selection server managing the selection as shown in Figure 2 A. The reachability information is sent from the devices or from management proxies in the gateways to the management server. The constraint information is collected

from the gateways and sent to the management server. The policy information is updated from the management interface to the management server. The centralized server runs the gateway selection algorithm creating a target distribution. Changes in the target distribution are sent via the management proxies to the concerned devices.

Since all the information is centrally available, there are no convergence problems in the selection process. However, the management server acts as a single point of failure and the capacity of the management server limits the scalability of the network. This is the major drawback in this design, although it could be solved by redundant servers.

2) Distributed: In the *distributed* alternative, the selection of the best candidate is done by the gateways themselves as shown in Figure 2 B. In this approach, the gateways share the reachability, constraint and policy information among them and every gateway runs the same gateway selection algorithm to choose the right candidate.

The main challenge of this alternative is the communication needed between the gateways to synchronize the information. To carry out this communication, either the cellular links or the capillary network itself must be used. Both alternatives have limited capacity. Moreover, because of the time required by replication, the information at the gateways might be inconsistent at some points in time. That inconsistency might cause devices to get inconsistent commands from different gateways during short periods of time.

In the case of large domains, the communication should be limited to a minimum set of gateways to reduce delays. Thus, gateways replicate information only to other nearby gateways. Alternatively, one gateway could be established as the decision-making node. This approach would be very similar to the *centralized* approach above. This reduces the time for replicating the information but, instead, the system would become less robust due to having a single point of failure.

3) Autonomous devices: In the *autonomous devices* alternative, the IoT devices make their own decision to which gateway they connect. Figure 2 C shows this model. The reachability information is then produced and utilized locally by the devices. The constraint and policy information is requested by every single IoT device from the gateways or from particular servers in the network. In the latter case, the IoT device must indicate to a constraint server the list of potential gateways, and the constraint server replies with the constraints of the aforementioned gateways.

One of the challenges with this approach is the complexity required at the devices to run the gateway selection algorithm. Typical IoT devices have limited processing power and memory, and executing the algorithm consumes the device's battery. Thus, this approach would be limited to IoT devices with enough capacity or, in some cases, used in a hybrid approach where the limited devices use a *centralized* approach instead. Furthermore, it is difficult to implement load distribution in this approach, since there is no coordination between the IoT devices. Oscillation easily appears. For example, a gateway with low load may cause a race of devices connecting to it, suddenly increasing the load above the average.

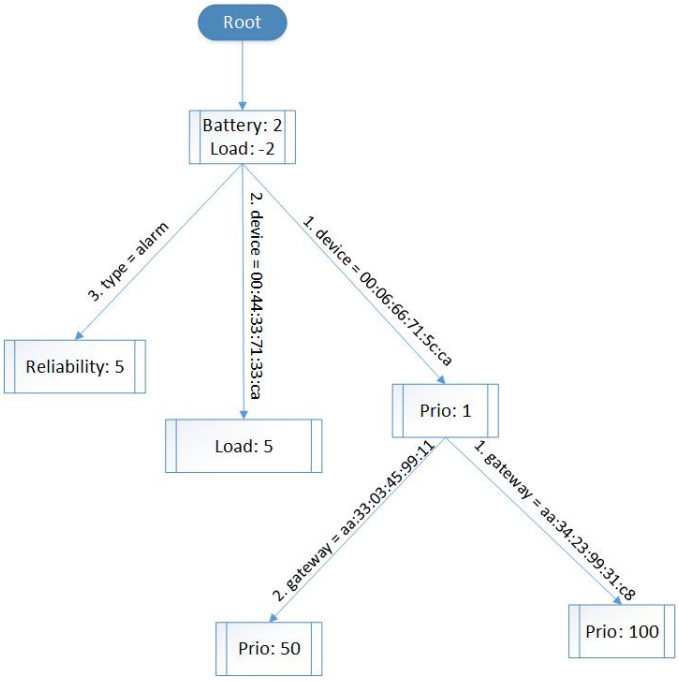

Fig. 3: Policy Tree

III. Gateway Selection Process

This section describes how the *gateway selection process* works and how the best candidate is chosen according to the gateway selection parameters. First, we describe how to model the gateway selection parameters within the gateway selection algorithm. Afterwards, we present the gateway selection algorithm itself and provide an implementation example.

A. Algorithm Input

The *gateway selection algorithm* takes the policy, the constraints and the reachability information as input in order to produce a selection of gateway for each device as output.

1) Policies: To enable expressive policies, the policies are described using a decision tree. A node in the tree can have ordered branches toward other nodes as depicted in Figure 3. Each branch carries a condition determining when the branch is traversed. Processing starts from the root and traverses the first of the ordered branches for which the condition is fulfilled. The condition can specify that a branch is to be traversed for a given (1) device, (2) gateway, (3) interface, or (4) node type. Once no other branches can be traversed, the policy description of that node is applied. The policy description is a list of constraint names and the corresponding weights of those constraints. Additionally, a constant value can be given with a special constraint named "priority". Using this simple model, a versatile range of policies can be described. The policy tree can be described using JavaScript Object Notation (JSON) or eXtensible Markup Language (XML). The former is used in our implementation.

2) Reachability: Reachability information is collected from devices and/or gateways. Based on the reachability information, a full or partial reachability graph is built. The reachability graph R has a link (g, d) between gateway g and

device d if either g has reported reachability to d, or d has reported reachability to g. A timer is attached to each link in the reachability graph. Reachability links that have not been refreshed will be deleted from the graph after a time out.

3) Constraints: Constraints are key-value pairs describing the properties of a gateway. The key is the constraint name c and the value is a numeric value $v_c(g)$ of the constraint for a specific gateway g.

B. Gateway Selection Algorithm

The algorithm operates by iterating through all devices to find a target gateway for each device. In order to maintain stability, the devices are iterated in the order of joining the network. Thus, a recently joined device does not cause changes in the allocation of the existing devices, but rather a suitable gateway is determined for the newly joined device instead.

To find the gateway for a device d, a preference value $p(g)$ is calculated for each gateway g that is in the set of reachable gateways $G = \{g : \exists(g, d) \in R\}$. For the specific combination of gateways and devices, a node in the policy tree is obtained as described in the previous subsection. The policy node defines a set of constraints C with a weight w_c for each constraint $c \in C$. Additionally, a constant value k may be defined (otherwise $k = 0$). The constant value k is defined in Section III-A as "priority". The preference of a gateway g is calculated by weighting the constraint value $v_c(g)$ with the corresponding weight w_c:

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

The gateway with the highest preference is selected for the device d.

$$g_{sel}(d) = \underset{g \in G}{\operatorname{argmax}}\, p(g) \tag{2}$$

Additionally, for the purpose of load distribution, a particular constraint $c_{conn}(g)$ is dynamically defined for each gateway g. This constraint reflects the number of devices allocated to a gateway during a particular point in the algorithm. At the start of the algorithm, the value is reset to $c_{conn}(g) = 0$, $\forall g \in G$. Each time a node is allocated to a gateway g the constraint is increased: $c_{conn}(g) = c_{conn}(g)+1$. The constraint is used in the policy and the preference calculations like other constraints.

C. Example

Let us use an example to illustrate the use of policies to obtain different selection results. The example policy description in Figure 4 contains a generic policy (lines 11-12), an exception policy for a specific device (lines 2-7), and an exception policy for a given device type (lines 8-10). The conditions for selecting the exception policies, i.e. the branches in the policy tree, are specified on lines 2 and 8, respectively.

The generic policy gives the same weight for considering both battery and load, but with different sign. Thus, the preference of a gateway increases when the battery level of the gateway increases, while the preference decreases when the load level increases. A gateway A with constraints {"load": 1, "battery": 5} gets a preference of $-2 * 1 + 2 * 5 = 8$, while a gateway B with constraints {"load": 3, "battery": 4} gets a lower preference of $-2 * 3 + 2 * 4 = 2$. Thus, gateway A is selected.

```
1:  {
2:    "device=00:06:66:71:5c:ca": {
3:          "gateway=aa:34:23:99:31:c8": {
4:                "priority": 100
5:          },
6:          "priority": 1
7:    },
8:    "type=alarm": {
9:          "reliability": 5
10:   },
11:   "battery": 2,
12:   "load": -2,
13: }
```

Fig. 4: Policy in JSON Format

The type-specific policy defines a branch for devices of the type "alarm". The node at the branch instructs that alarm devices should be connected to the gateway with the highest reliability constraint.

The device-specific policy is defined by a branch for a given device identity. Within the branch, a particular gateway is defined (using a branch in the policy tree) to have a priority of 100 while all other gateways have the priority of 1; thus, the device is allocated to the indicated gateway unless it is unavailable. The special constraint "priority" is used to specify constant values.

IV. Implementation

During our implementation phase, both the *distributed* and the *centralized* architecture were implemented. The *autonomous* architecture was deemed impractical for constrained devices due to its requirements on processing power and memory, the usage of which were already close to the limits.

This section describes the implementation from a technical viewpoint while the next section will explain the reason behind each choice made.

A. Implementation of Distributed Architecture

In the *distributed* architecture, gateways share their constraints and reachability information. Each gateway implements the gateway selection algorithm in a replicated way based on the information. The information must be synchronized between gateways to ensure that all gateways obtain the same result.

Since gateways need to share information, they need to have connectivity to each other. This could be achieved using the cellular uplink, the capillary network itself, or through a separate network. Connecting all gateways with each other in a mesh would not be scalable, therefore the network was split into gateway selection domains. Within each domain, gateways share information and devices can only move between the

gateways in the domain. [1] In our prototype implementation, we connected gateways with Ethernet for simplicity.

For distributing information between the gateways, we extended the Open Shortest Path First (OSPF) [8] routing protocol with custom link state advertisements. When a constraint value of a gateway changes, this gateway advertises the new constraint value to all other gateways over OSPF. To reduce the backhaul traffic caused by downloading policies from the management server, the gateway with the highest OSPF identifier is selected as the master. The master periodically downloads the policy and distributes any changes between the gateways. The policy is provided using a HTTP REST [9] interface to the gateways. Consequently, the management server only has a minimal role in supporting gateway selection by providing the policy. [2]

In this implementation, TMote Sky [10] devices were used as sensors. They run 6LoWPAN [11] over an IEEE 802.15.4 interface. We based our device implementation on the Contiki operating system, with the RPL (IPv6 Routing Protocol for Low-Power and Lossy Networks) [12] routing protocol between the gateway and the device. RPL implements the messaging for allowing the gateway and the device to detect each other. At the application layer devices implement two protocols: Constrained Application Protocol (CoAP) [13] for data communication, and Lightweight M2M Protocol (LWM2M) [14] for device management part. LWM2M objects at the device represent the reachable gateways, the target gateway selection and the currently connected gateway.

The IEEE 802.15.4 interface of the gateway is implemented by using a USB-connected TMote Sky device, which operates as the root for RPL. RPL uses a Destination-Oriented Directed Acyclic Graph Identifier (DODAGID) to identify the RPL root. The root device provides a HTTP REST interface [9] toward the software in the gateway. This REST interface allows the gateway software to query the root about devices in the network that the root can reach. Reachability information is thus generated by the gateway. Reachability in this sense is defined as the existence of a RPL route between the device and the gateway.

When a device joins the network, it first selects a random root device, i.e. a random gateway. The gateway detects the existence of this device by periodically querying the root. Each gateway advertises the connected devices over a custom OSPF link state advertisement to other gateways. Consequently, every gateway learns about all devices in the network.

Each time there is a change in reachability, policy or constraint information obtained via OSPF, the gateway selection algorithm is restarted. As OSPF quickly floods changes between gateways, all gateways use the same information and starts the algorithm roughly at the same time. It is important that all gateways processes the devices in the same order so that all gateways obtain the same result. Implementing the join order, as described in Section III-B, is difficult because of the need to synchronize clocks. Therefore, devices are ordered according to their MAC address instead.

[1] In the case gateway selection is needed between the domains, a hierarchical solution combining distributed and centralized architectures can be used.

[2] The management server may additionally be responsible for bootstrapping, registration, access control, configuration, and other management functions for both devices and gateways. These are out of the scope of this paper.

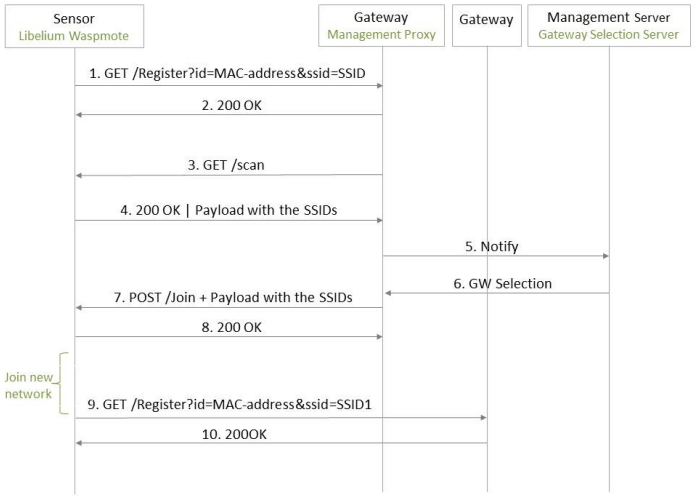

Fig. 5: Gateway Selection Sequence Diagram for the centralized architecture

To make the device switch a gateway, the gateway to which the device currently is connected sends a command to the device. The command is implemented as a LWM2M like command transported over CoAP. The command contains the DODAGID of the target gateway. Additionally, the command can include a list of alternative gateways to be used in case the target gateway becomes unavailable.

B. Implementation of Centralized Architecture

The selection of the gateway in the *centralized* architecture is controlled by a central node: the *gateway selection server*. Implementation-wise, the *gateway selection server* is either integrated as a part of the management server or implemented as a separate component connected to the management server. In our implementation, the gateway selection server is part of the management server. Additionally, the management server is connected to a *management interface*, which provides a dashboard for managing the IoT network. The management interface graphically presents the gateways devices connected to it and allows editing the policy.

The main function of the gateway selection server is to collect the reachability, constraints and policy information as input and generate the target allocation of devices to gateways. Our server exposes a HTTP REST interface for the gateways to post updated constraint information. Constraints can also be edited from the management interface, which is useful for testing and demos. The policy is locally stored on the gateway selection server and updated by the management interface through a set of HTTP REST commands. Both policies and constraints are described in JSON format.

In the centralized implementation, two types of devices are supported: IEEE 802.15.4 based devices with RPL routing and IEEE 802.11b/g based devices. Depending on the device type, the reachability information can be obtained either from gateways or from the devices.

The first case is utilized on TMote Sky devices operating over IEEE 802.15.4 radio. The devices run RPL [12] to create routes toward the RPL root, and the root provides a REST interface for the gateway to obtain the list of the neighbours reachable from the root. The gateway posts changes in the reachability information to the gateway selection server.

The second case is used in Libelium Wasmote devices [15] with IEEE 802.11b/g [5] wireless capabilities. Libelium Waspmote devices provide a device centric view of the reachability. The Waspmote performs a scan over all channels to find the SSIDs of the surrounding gateways. This scan is initiated by a command from the server. The reachability information is posted via the gateway to the gateway selection server.

Our gateways support several interfaces per gateway. Thus, a gateway can have both IEEE 802.15.4 and IEEE 802.11b/g interfaces, and even several interfaces of the same type. For IEEE 802.15.4, the interface is identified by the Destination-Oriented Directed Acyclic Graph Identifier (DODAGID), and in IEEE 802.11b/g the interface is identified by a Service Set Identifier (SSID). Consequently, reachability and gateway selection commands contain these interface identifiers.

After all the input information is gathered, the *gateway selection server* finds out the best gateway interface for every device and sends the interface identifier information to the affected devices. Messages are forwarded via the gateway in which the device currently is connected to. In our implementation, different protocols are needed to send the command to different device types. The TMote Sky supports CoAP, and the implementation works like in the distributed implementation. Since the Waspmote do not support IPv6 and CoAP, we implemented a HTTP REST interface to the device instead. In a final implementation, without the above restrictions, the Lightweight Machine to Machine (LWM2M) [14] protocol would be used instead.

The management of the devices from the *gateway selection server* could be implemented with transparent gateways, allowing management commands to be sent to devices directly from the management server to the device. However, in a practical implementation, a few issues suggest involving the gateways as a proxy on the management path. Firstly, the reachability information within the IEEE 802.15.4 wireless network needs to be collected from the gateway perspective. That implies the gateway needs to act as a LWM2M client as well. Secondly, in the case of NAT [16] between the Management Server and the devices, a proxy in the gateway simplifies the tunnelling of inbound commands over an outbound connection. Thirdly, Libelium Wasmote devices [15] do not support CoAP [13]. The LWM2M protocol is transported on top of CoAP. That means that a conversion between CoAP and HTML was needed and the gateways are the best candidates due to their processing power. All these reasons made us decide to implement a management proxy in each gateway acting as a back-to-back server and client.

Figure 5 shows the interaction of the different elements involved in sharing the reachability information between the Libelium Waspmote devices and the Management Server component. First of all, the Libelium Waspmote device automatically connects to a network. Once the device has connected, it will send a registration message (1) to a gateway in which its address is predefined in the sensor's EEPROM. That information will contain the device's MAC address and the SSID of the network that the sensor is connected to. After the sensor is registered, the gateway will ask (3) the sensor to scan all the available access points (AP) around it. Then, the Libelium Waspmote will perform an active probe scan of access points in all the 13 channels and returns (4) to the gateway the MAC address, signal strength, SSID name, and security mode of the found access points. The scan command output [17] format is:

Channel	RSSI	Security	WPA Conf	WPS	MAC Address	SSID

The gateways will forward that information to the management server. The management server will read the list of reachable gateways and send a command (6,7) to the device. That command will include in the payload a list of target gateways in priority order to allow for backups in case of failure of the target gateway. The sensor will try to connect to the first option or, subsequently, to the consecutive options if that fails. Once the device changes gateway, the registration message (9,10) is triggered again.

V. Evaluation

In this section, we evaluate the three architecture options and provide some insights that help to clarify our preference for the centralized solution.

Figure 6 shows the message paths for different types of operations in the three architectures. The figure focuses on messages over the two wireless interfaces: the mobile backhaul and the capillary network. The figure assumes that $Device_1$ and $Device_2$ are reachable from $Gateway_2$ while $Gateway_1$ cannot reach any device.

Policies are distributed infrequently as they are rather static. In the centralized architecture the new policy is not transported over any wireless interface. In the distributed architecture, the updated policy must be sent to each gateway separately. In the autonomous architecture, the new policy must be sent to all gateways, which forward it to all reachable devices.

In the centralized architecture, it is sufficient to transport the updated constraints of a gateway to the management server. In the distributed architecture, the constraints must be sent to all gateways over the mobile network. The transmission between a pair of gateways requires one message in the uplink direction and one in the downlink direction. In the autonomous architecture, the gateway sends its updated constraint to all reachable devices.

Reachability updates are frequent if devices are mobile. Updated reachability information generated by a device must traverse both the capillary and the mobile networks to reach the management server in the centralized architecture. In the distributed architecture, the message must be flooded between gateways, each transmission passing over the mobile network twice. In the autonomous architecture, the reachability information is only used locally by the device.

Each time a parameter changes, the target gateway of one or more devices may change. In the centralized architecture, a command to switch gateway is sent over the two wireless interfaces to the affected devices. In the distributed architecture, each gateway locally performs the gateway selection

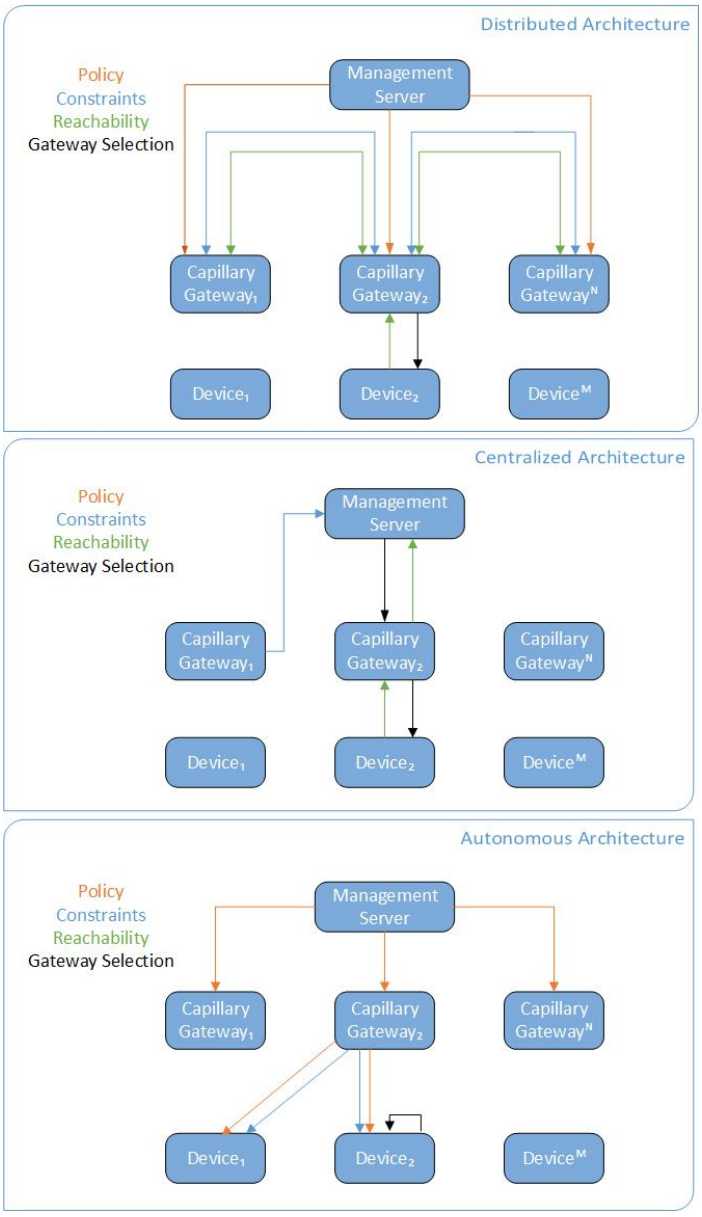

Fig. 6: Messages used in the evaluation

algorithm. Messages are needed to be sent only to the affected devices over the capillary network interface. In the autonomous architecture, the device performs the calculation locally and switches gateway without any messaging.

	Distributed		*Centralized*		*Autonomous*	
	Cell	*Cap*	*Cell*	*Cap*	*Cell*	*Cap*
Policy	N	0	0	0	N	M
Constraints	$2N(N-1)$	0	1	0	0	D
Reachability	$2N(N-1)$	1	1	1	0	0
GW Selection	0	1	1	1	0	0

TABLE I: Number of messages per operation

Table I shows an analysis of the traffic required for each of the above operations. To make the comparison more generic, we count the number of HTTP REST requests instead of the number of bytes or messages. Thus, we do not consider message size, connection setup, acknowledgement, fragmentation and packet loss. We separately consider messages in the cellular uplink (abbreviated as Cell) and the capillary network (abbreviated Cap). In the distributed solution, we assume that the gateways communicate with each other via the cellular network, so that each request traverses the cellular network twice. We denote the number of gateways with N and the number of devices with M. The average number of devices within reach of a gateway is denoted D, which typically is $D \propto M/N$. For flooding in OSPF, we assume full connectivity, thus a message from one gateway is sent to $N-1$ other gateways. Sending a message to all devices under a gateway is assumed to be implemented with unicast using HTTP REST, rather than broadcast.

Based on Table I, we can analyse the total traffic generated for different scenarios. The traffic is dependent on the frequencies of updates in the gateway selection parameters. We denote the frequency of policy updates with f_{pol}, the frequency of constraint changes per gateway with f_{const}, and the frequency of reachability updates per device with f_{reach}. The frequency of gateway selection commands, f_{sel}, is proportional to the total number of updates $f_{pol} + Mf_{reach} + Nf_{const}$ as each update may, with a given probability, cause a device to be allocated to another gateway. In our analysis, a selection command is sent once per 10 updates, but the choice of value turned out to have only a minor effect on traffic. D denotes the average number of devices within reach of a gateway. We assume every devices can be reach from three different gateways. The default values of the parameters are given in Table II.

Parameter	*Default value*
f_{pol}	1 update / 24 h
f_{const}	1 update / 1 h
f_{reach}	1 update / 10 min
f_{sel}	$(f_{pol} + Mf_{reach} + Nf_{const})/10$
D	$3M/N$

TABLE II: Default values of parameters in the analysis

Figure 7 shows the total number of requests in the network for a network with 10000 devices and 1000 gateways. Figure 8 shows the same scenario for a network, where the same 10000 devices are distributed between 100 gateways. In both figures, the frequency of reachability updates f_{reach} varies between one update per 1 minute and one update per 24 hours. This correspond to different degree of device mobility. We see that the distributed architecture generates several magnitudes higher traffic than the other architectures. The centralized architecture generates least traffic, except when mobility is high, when the autonomous architecture performs better. In Figure 9 the frequency of constraint updates f_{reach} varies between one update per 1 minute and one update per 24 hours. This mainly affects the traffic in the autonomous architecture. We noticed that varying the other parameters only has a minor effect on the traffic. The last table, Table III, evaluates the load on the management server. In all evaluated cases, the centralized architecture showed the highest load on the server.

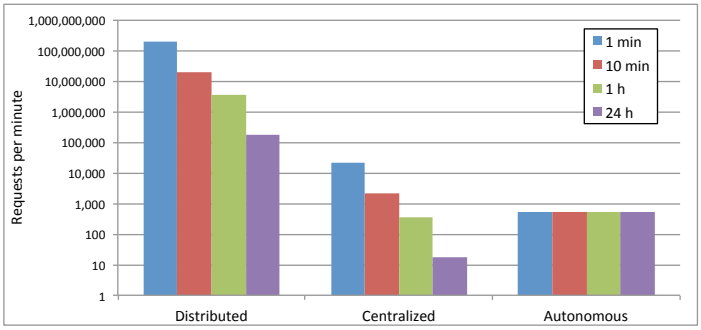

Fig. 7: Requests for various reachability update frequencies, 100 gateways

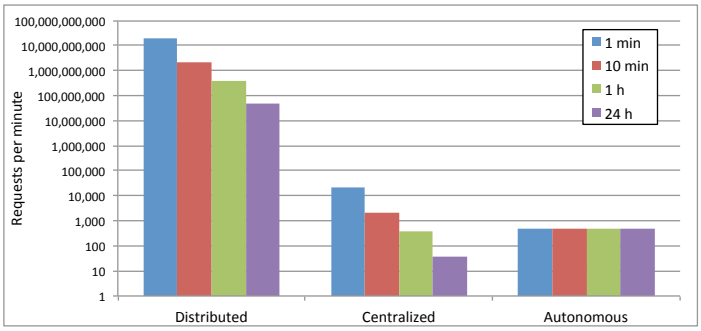

Fig. 8: Requests for various reachability update frequencies, 1000 gateways

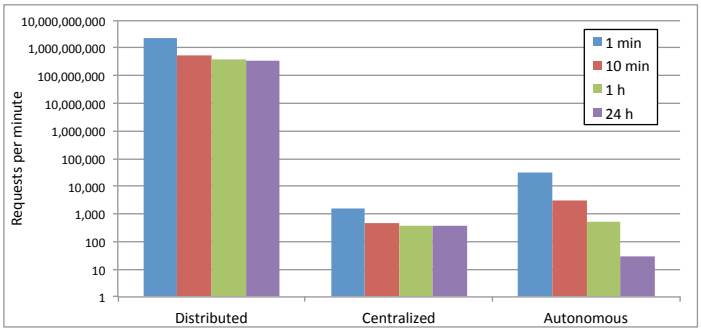

Fig. 9: Requests for various constraint update frequencies, 1000 gateways

VI. Conclusion

In this paper, we addressed the problem of gateway selection in a capillary network. Without any mechanism, a device typically selects the gateways randomly, without considering the uplink capacity or the properties of the gateways. Gateway selection allows devices to make informed selection in order to globally optimize the network. The paper presents an overview of different gateway selection solutions for capillary networks, some of which we implemented. The solutions mainly differ in the location where the selection decision is made. According to our analysis, a distributed architecture creates a high amount of traffic in the cellular network. The centralized architecture has low traffic both in the capillary and the cellular network but the load on the management server is high. The autonomous devices make the network simple with low traffic and low load on the server, but the device itself has a high complexity as it must perform the gateway selection. This complexity is often too demanding for the current sensors. Our implementation utilizes flexible policies, which takes into account multiple factors to select the optimal gateway. With automatic gateway selection, load can be distributed between gateways, while considering gateway properties, such as current load. This allows a better utilization of the network resources and optimization of the end-to-end path through both networks.

	Load on Server
Distributed	Nf_{pol}
Centralized	$Nf_{const} + Mf_{reach} + Mf_{sel}$
Autonomous	Nf_{pol}

TABLE III: Load on the Management Server

Acknowledgment

The authors would like to thank Tero Kauppinen, Miika Komu and Mert Ocak for their contribution in the implementation of the prototypes and Petri Jokela for his feedback.

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. [Online]. Available: http://dx.doi.org/10.1016/j.future.2013.01.010

[2] O. Mazhelis, H. Warma, S. Leminen, P. Ahokangas, P. Pussinen, M. Rajahonka, R. Siuruainen, H. Okkonen, A. Shveykovskiy, and J. Myllykoski, "IoT SOTA report 2013 - Internet-of-Things Market, Value Networks, and Business Models: State of the Art Report," Jan. 2013. [Online]. Available: http://www.internetofthings.fi/

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

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

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

[6] S. Singh and K.-L. Huang, "A robust m2m gateway for effective integration of capillary and 3gpp networks," ser. Advanced Networks and Telecommunication Systems (ANTS), S. Singh and K.-L. Huang, Eds. 2011 IEEE 5th International Conference, 2011, pp. 1–3.

[7] V. Misic, J. Misic, X. Lin, and D. Nerandzic, "Capillary machine-to-machine communications: The road ahead," in *Ad-hoc, Mobile, and Wireless Networks*, ser. Lecture Notes in Computer Science, X.-Y. Li, S. Papavassiliou, and S. Ruehrup, Eds. Springer Berlin Heidelberg, 2012, vol. 7363, pp. 413–423.

[8] *RFC 5340; OSPF for IPv6*, IETF.

[9] L. Richardson and S. Ruby, *RESTful Web Services*. O'Reilly Media, 2008.

[10] "TMote Sky Datasheet." [Online]. Available: http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf

[11] *RFC 6282; Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks*, IETF.

[12] *RFC 6550; RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*, IETF.

[13] *RFC 7252; The Constrained Application Protocol (CoAP)*, IETF.

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

[15] "Libelium webpage." [Online]. Available: http://www.libelium.com/

[16] *RFC 2663; IP Network Address Translator (NAT) Terminology and Considerations*, IETF.

[17] "Microchip, Wifly Command Reference." [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/50002230A.pdf