Wireless Personal Communications manuscript No.
(will be inserted by the editor)

Sensor function virtualization to support distributed intelligence in the Internet of Things

Floris Van den Abeele · Jeroen Hoebeke · Girum Ketema Teklemariam · Ingrid Moerman · Piet Demeester

Received: date / Accepted: date

Abstract It is estimated that - by 2020 - 50 billion devices will be connected to the Internet. This number not only includes TVs, PCs, tablets and smartphones, but also billions of embedded sensors that will make up the "Internet of Things" and enable a whole new range of intelligent services in domains such as manufacturing, health, smart homes, logistics, etc. To some extent, intelligence such as data processing or access control can be placed on the devices themselves. Alternatively, functionalities can be outsourced to the cloud. In reality, there is no single solution that fits all needs. Cooperation between devices, intermediate infrastructures (local networks, access networks, global networks) and/or cloud systems is needed in order to optimally support IoT communication and IoT applications. Through distributed intelligence the right communication and processing functionality will be available at the right place. The first part of this paper motivates the need for such distributed intelligence based on shortcomings in typical IoT systems. The second part focuses on the concept of Sensor Function Virtualization, a potential enabler for distributed intelligence, and presents solutions on how to realize it.

Keywords Internet of Things · Distributed systems · Gateway · Cloud · WSN · WSAN · 6LoWPAN · CoAP · Sensor Function Virtualization · Distributed Intelligence

1 Introduction

Today, most of the data available on the Internet is generated by humans. With more and more everyday objects or sensors being connected to the Internet, the amount of data generated by things is going to increase rapidly. It is

Department of Information Technology (INTEC), Ghent University - iMinds
Gaston Crommenlaan 8 box 201
B-9050 Ghent, Belgium
E-mail: {fvdabeele, jhoebeke, gketema, imoerman, pietdm}@intec.UGent.be

estimated that by 2020, 50 billion devices will be connected to the Internet, outnumbering the number of human beings on our planet [6]. This vision is commonly referred to as the Internet of Things.

In the Internet of Things, heterogeneous objects will grasp information about our physical world and inject it in the virtual world where it can be used as input to all kinds of services. Depending on the type of service, a decision can be taken to act again upon the physical world. As a result, everything around us will become an integral part of the Internet, capable of generating and consuming information. It is evident that the Internet of Things may have a great impact in a wide range of application domains such as health, manufacturing, building automation, transportation, smart cities, logistics, etc.

In order to connect things to the Internet, they need to be equipped with processing and communication capabilities. In many cases these capabilities are very limited, as these devices may need to run on batteries for several months or years or need to be produced at an extremely low cost. Such devices are often referred to as constrained devices, many of them belonging to Class 1 (approximately 10KiB RAM and 100KiB ROM) as defined by the IETF lwig working group [1]. Specific low data rate and low power communication technologies have been designed, such as IEEE 802.15.4. Further, as typical Internet protocols had not been designed for such small footprints, initial efforts to interconnect these devices to the Internet resulted in proprietary protocols and architectures. However, their incompatibility with widely adopted Internet protocols has hampered their uptake and the realization of the IoT vision.

The last few years, this mindset has been changing and many efforts have been put into the extension of Internet technologies to constrained devices. Most noteworthy are the efforts of the Internet Engineering Task Force, the de facto standardization organization for Internet protocols. Their initial efforts focused on the networking layer and resulted in the integration of constrained devices and constrained networks in the IPv6 Internet. As a next step, they now target the efficient integration of these devices in web services. So far, the IETF has standardized the Constrained Application Protocol (CoAP), which can be seen as an embedded counterpart of HTTP. With these technologies, it has become possible to interconnect tiny objects or networks of such objects with the IPv6 Internet and to build applications that interact with them using embedded web service technology, bringing us one step closer to the realization of the Internet of Things.

From the above description, it becomes clear that it has been a challenge to fit an Internet-compatible protocol stack on devices with very limited capabilities. It required the careful design of tailored communication protocols. One can question how far one can go to further extend these devices with additional functionalities or intelligence typically encountered in more powerful devices, such as access control, preprocessing of data, data formatting, resource visibility, etc. At some point it will become technically infeasible to put additional intelligence on the devices themselves due to their constraints. Consequently, such intelligence needs to be outsourced to more powerful devices such as gateways, routers, neighboring devices, the cloud, etc. In addition, as every IoT application may have different requirements and devices may be very heterogeneous, there will be no single recipe on how to optimally distribute this intelligence.

The optimal placement of functionalities or intelligence is only one aspect of the whole picture. Current networks and radio technologies are very homogeneous, homogeneous in a sense that the resulting communication infrastructure often offers the very same service to every application running on top of it. Looking at the large number of IoT application domains or the variety of IoT applications with heterogeneous requirements within a single application domain, optimal support of IoT applications also requires adaptations of the network behavior. Whenever possible, networking elements should expose the necessary interfaces in order to optimally configure the underlying network behavior.

Combining both aspects brings us to the concept of distributed intelligence in the context of the Internet of Things: depending on the application, user and policy requirements, it is decided where to place the intelligence to operate certain functions and how to optimally configure the communication infrastructure.

In the remainder of the paper, we will first further motivate the need for such distributed intelligence by identifying a number of shortcomings in typical IoT systems in section 2. Section 3 presents Sensor Function Virtualization (SFV) as an important enabler for realizing distributed intelligence. Before moving on to examples of SFV, section 4 gives a short overview of the IETF IoT protocol stack. This stack is used for examples of SFV in the unconstrained and constrained domain in sections 5 and 6 respectively. Section 7 presents the related work that we have identified in the literature. The paper ends with a number of conclusions and future work in section 8.

2 The need for distributed intelligence

2.1 Generic IoT system

Figure 1 shows a high-level representation of a generic IoT system from a communication and processing point of view. On the left there are the embedded devices with some form of processing and communication capabilities. These devices can be very heterogeneous in terms of energy provisioning (energy harvesting, battery powered, mains powered), communication capabilities (IEEE 802.15.4, BLE, IEEE 802.11, etc.), processing power and communication behavior (always connected versus intermittently connected). Examples of such devices are a battery-operated environmental sensor, a Wi-Fi weighing scale, a tracking device with a GPRS module or even a powerful smart phone. In an IoT system, these devices typically collect information about the physical world and, possibly after some (limited) local processing, communicate this information to an external service or another device for further processing.

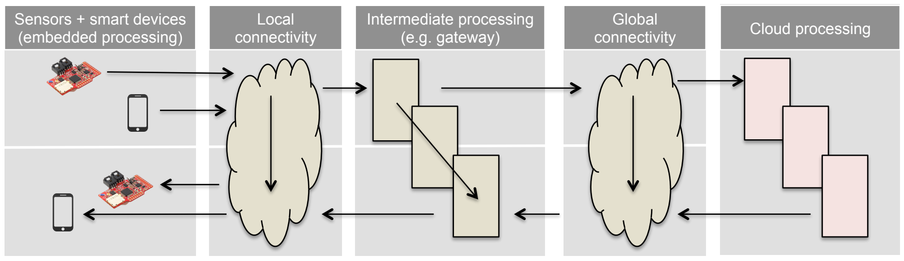

Sensors + smart devices (embedded processing)
Local connectivity
Intermediate processing (e.g. gateway)
Global connectivity
Cloud processing

Fig. 1 A generic Internet of Things system

The information is then transmitted over a local communication network, such as a 6LoWPAN network or a WLAN network, in order to reach the Internet. On its way out, it passes intermediate processing components such as a home gateway, a border router or an access point. In most cases, such a component will mainly perform some protocol translations (e.g. conversion between 6LoWPAN and IPv6, NAT translation, etc.). After that, the information is communicated over a global communication network (i.e. the Internet) until it reaches a cloud service. The cloud service will process the sensed data, enrich it, combine it with other sources of information and eventually convert it into retrievable knowledge that needs to be stored or actions that need to be performed in the real world. In the latter case, there will be a flow from right to left in figure 1, until the action reaches an embedded device such as an actuator.

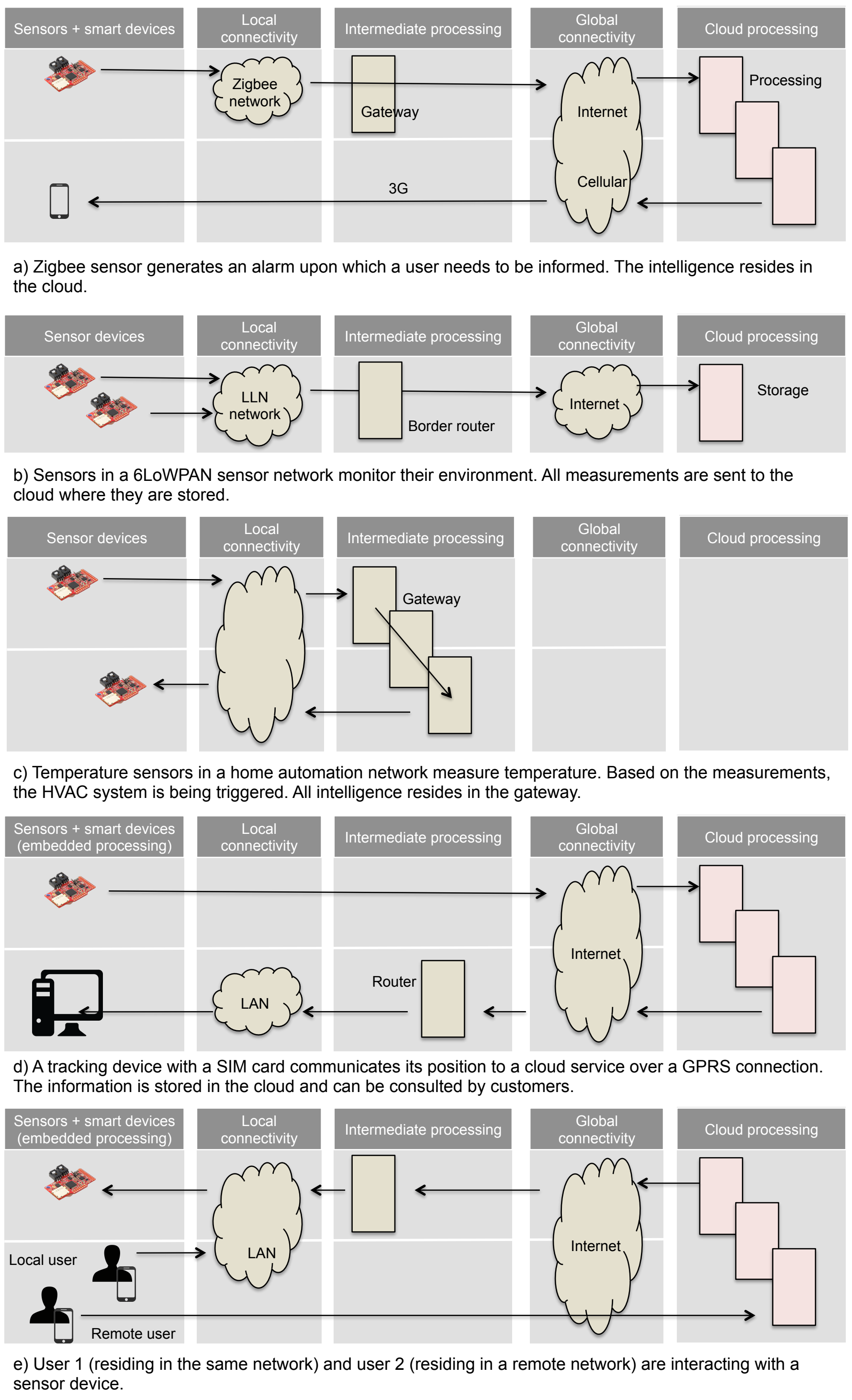
Of course, many variants of the above generic IoT system exist and in many IoT systems not all of the components shown in figure 1 need to be present or involved. Therefore, in figure 2, we have mapped a number of realistic IoT use cases to this generic system.

2.2 Open challenges

In this subsection we will illustrate through a number of concrete examples, i.e. instantiations of the generic IoT system shown in figure 1, some of the existing limitations of state-of-the-art IoT systems and pinpoint the root cause of these limitations.

2.2.1 Act in time

Many IoT business models adopt a cloud-based approach as partially illustrated in figure 2a. Sensor data is pushed to the cloud where it is being processed. If needed, corrective actions are taken. All intelligence is centralized in the cloud, which makes it convenient for the cloud service provider to maintain the system and to roll out new functionalities. However, depending on the application, the complete sensing and actuation cycle might have to be completed within a certain time interval. For instance, turning on a light using a



a) Zigbee sensor generates an alarm upon which a user needs to be informed. The intelligence resides in the cloud.

b) Sensors in a 6LoWPAN sensor network monitor their environment. All measurements are sent to the cloud where they are stored.

c) Temperature sensors in a home automation network measure temperature. Based on the measurements, the HVAC system is being triggered. All intelligence resides in the gateway.

d) A tracking device with a SIM card communicates its position to a cloud service over a GPRS connection. The information is stored in the cloud and can be consulted by customers.

e) User 1 (residing in the same network) and user 2 (residing in a remote network) are interacting with a sensor device.

Fig. 2 Five IoT scenarios mapped to the generic IoT system from figure 1

light switch requires a worst case latency of 200ms or better, whereas other home automation applications may live with higher latencies. Consequently, depending on the type of IoT application, the processing functionality may reside either far away from the constrained devices (in the cloud) or must reside nearby in order to meet certain performance requirements. Future IoT systems should be able to cope with these requirements, e.g. by supporting distributed processing that puts the intelligence wherever it is most needed.

Even when the processing functionality resides close to the sensors and actuators, e.g. on a gateway as shown in figure 2c, it may still be possible that the underlying network is not capable of meeting the performance requirements imposed by the IoT application e.g. due to suboptimal routing, inefficient medium access control or competition with other wireless traffic. To tackle this, it should not only become possible to place the intelligence where it is needed, but also to optimally configure the underlying network. Similar observations can be made for cases where for instance synchronization is needed.

2.2.2 Work offline

IoT applications that solely rely on intelligence in a cloud system will completely break upon an interruption of the Internet connectivity (the "global connectivity" in figures 1 and 2). For instance, one can create a building management system that is fully managed and controlled by the cloud based on locally collected sensor data. However, in case the cloud is unavailable, the building management system should still be able to deliver a minimal service level. Consequently, the core functionalities of an IoT application should reside in the local network. This way, using simplified local processing, it is still possible to have an operational application albeit with reduced functionality. If connectivity is available, more advanced functionalities, e.g. by using externally available data, can be offered to the users. Again, distribution of intelligence and processing is needed to offer a robust IoT system.

2.2.3 Serve many

IoT systems such as the one shown in figure 2e, may not only scale up to thousands of sensors and actuators, but may also involve a multitude of users, each user taking up a particular role in the overall system. For instance, in a building management system one can have the building owner, facility managers, tenants, cleaning staff, visitors, etc. Depending on their role and the corresponding policies, users should perceive a different system in terms of the data they can see, the actions they can take, etc. However, many of the involved devices are not capable of supporting this level of granularity in terms of visibility, access control, etc. due to their constraints. Even if they would be able to offer some of this functionality such as a security algorithm, it most likely would not scale with the number of users due to resource depletion, i.e. every additional user will require storage of additional state information. Therefore, it should become possible to outsource this functionality to more powerful infrastructure, preferably without impacting the constrained devices. Further, the outsourced functionality ought to be placed at the most optimal location. Note that this depends on the actual location of the user, e.g. a local versus a remote user.

2.2.4 Move and sleep

The IoT system shown in figure 2d involves tracking devices. These devices are mobile as they move with the object they are tracking. In most cases, as these devices are also battery powered, they are not permanently connected to the Internet. This has two consequences. First of all, the IP address of the device will change over time. Secondly, when users want to interact with the device, e.g. to perform some reconfigurations, the device will mostly be offline. Both aspects complicate the interaction with these devices for end users or external systems. Additional functionality has to be provided along the communication path in order to hide the sleepy and mobile behavior of the devices, thereby offering a uniform view to the outside world.

2.2.5 Monoglot

Most constrained devices do not posses the capabilities for supporting a wide range of data formats or protocols. In many cases, they only speak a single protocol and deliver their data in a particular format or content type. Further, the transferred data is often kept as compact as possible in order to reduce the communication overhead and to save energy. Lengthy, verbose descriptions are out of the question. When an IoT application is implemented as a vertical silo, where devices and cloud are designed to be interoperable, this may not be a problem. However, in more open systems where a variety of devices and services may interact with each other or where IoT devices can be connected to any service provider, this may easily lead to interoperability problems.

Different parties will most likely support different data formats (e.g. JSON versus XML, Fahrenheit versus Centigrade, etc.). When they need to interact, additional intelligence is needed for translating between incompatible data formats, else lack of interoperability will prevent their collaboration. Sometimes, the data generated by constrained devices is not self-descriptive. Therefore, on its path towards e.g. a cloud service, the data may be enriched with additional information to make it completely self-describing. A last example relates to the semantic web. Before semantic reasoning can take place, the sensor data needs to be tagged with additional semantic information. Again, placing such semantic descriptions on the constrained devices themselves might be too complex, so this functionality is better outsourced to more powerful devices. These examples illustrate that by dynamically placing intelligence in network or processing elements, interoperability can be greatly increased.

2.3 Distributed intelligence

From the previous examples it becomes clear that in order to optimally support a wide variety of IoT applications and user needs, additional intelligence is needed. This intelligence is not only related to the processing of data, but is also related to security, Quality of Service, network configuration, etc. Further, there is no single place where this intelligence has to be placed or activated. Depending on the situation, it may be spread from the devices themselves up to the cloud, covering all components in the chain shown in figure 1. In many cases, the intelligence needs to be distributed over different locations in order to deliver the desired functionality or performance. Further, it involves both processing and networking elements.

This is what we define as distributed intelligence. It is the cooperation between devices, intermediate communication infrastructures (local networks, access networks, global networks) and/or cloud systems in order to optimally support IoT communication and IoT applications. Starting from the application requirements, user needs and policies, intelligence is optimally distributed and activated in order to operate functions such as processing, security, QoS and to configure the communication infrastructure. Through distributed intelligence, the right communication and processing functionality will be available at the right place and at the right time as is illustrated in figure 3.

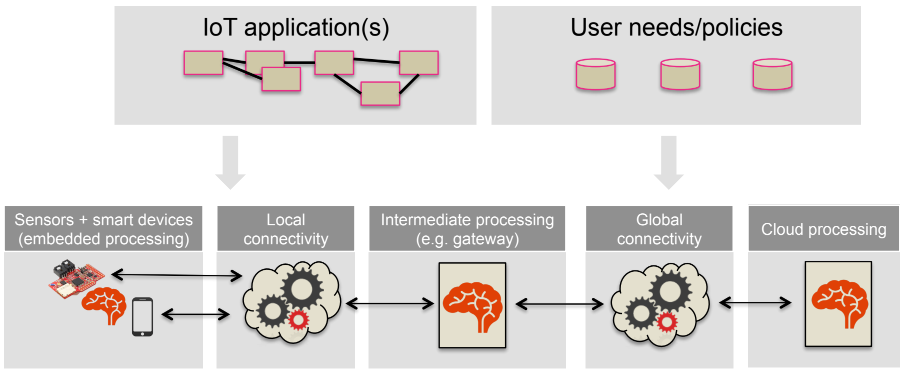


Fig. 3 The concept of distributed intelligence

Distributed intelligence will enable us to tackle the aforementioned shortcomings of existing IoT systems. It can contribute to increased interoperability, better usage of scarce resources, better application performance, improved user experience, more secure systems, etc. As can be seen from the previous discussion its realization is not straightforward and imposes significant challenges, mainly due to the fact that it is distributed in nature, involves both processing and communication and, communication-wise, pertains to multiple layers in the communication stack.

In the remainder of the paper, we will propose some concrete enablers to support the concept of distributed intelligence and will illustrate how they can work in the context of open IETF-based IoT protocol stacks. Hereby, we focus on the fact that typical IoT devices are constrained in nature and not capable of offering all functionality needed. Therefore, solutions are needed to outsource such functionalities to more powerful components, i.e. to virtualize this functionality as will be explained in the following section. Other interesting aspects of distributed intelligence, such as enabling an optimal configuration of the underlying network are outside the scope of this work.

3 Sensor Function Virtualization for the Internet of Things

The previous section illustrated why distributed processing is needed in the Internet of Things and gave a high-level overview of distributed intelligence. This section looks at how distributed processing can be realized while keeping in mind the challenges specific to the IoT domain. To this end, we propose an approach that enables distributed processing by offloading certain functionality from constrained devices to unconstrained infrastructure such as a (virtualized) gateway, the cloud and other (in-network) infrastructure. Hence the term “Sensor Function Virtualization” (SFV), where a sensor is defined more broadly to also include actuators and other types of constrained devices.

As the Internet of Things is expected to include up to 50 billion devices by 2020, any proposed solution for such sensor function virtualization should be able to scale as the number of devices increases. Here, two important points are to be noted. By running (parts of the) sensor function virtualization on cloud infrastructure, we expect our solution to profit from the elasticity provided by these environments: i.e. as the number of devices increases, more resources are automatically allocated by the cloud infrastructure to handle the increased load. Elasticity is a huge benefit for scalability that results from pooling resources in large data centers. Apart from elasticity, sensor function virtualization solutions should also follow a tiered design. Here, multiple tiers work together by each taking care of a part of the sensor function virtualization. Next to a more scalable approach, a tiered design also allows to mitigate some of the issues present in sensor function virtualization that relies solely on cloud-based infrastructure (e.g. high latency and non-functioning devices when Internet is unavailable).

Another important aspect for sensor function virtualization is the heterogeneity of both constrained devices and infrastructure. Constrained devices might be battery powered, might be part of a fixed communication infrastructure, might be mobile, have a large diversity in processing power and so on. Also, different forms of infrastructure that can aid in sensor function virtualization are expected to exist. In some deployments (e.g. typical WSN scenarios) there might be a mains-powered gateway that can assist in sensor function virtualization. In other settings, such gateways are not common but the access network itself might assist in sensor function virtualization (e.g. mo-

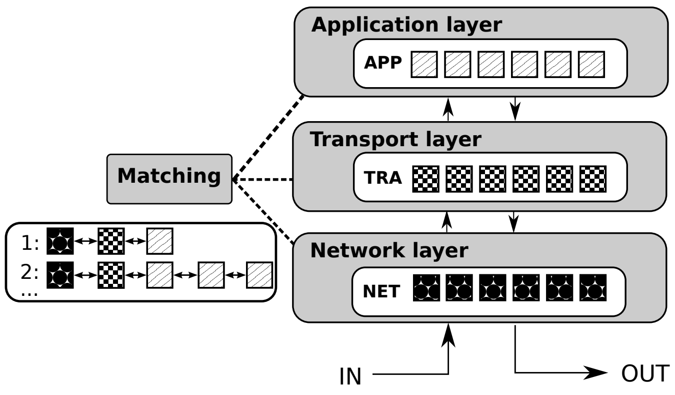


Fig. 4 Architecture for sensor function virtualization in the Internet of Things

bile devices that rely on GPRS communication). Finally, some environments might not provide any opportunities for tiering at all and here one is limited to external infrastructure (such as the cloud). Solutions for SFV have to be flexible in order to deal with these forms of heterogeneity and should try to shield users as much as possible from the heterogeneity of the underlying devices and infrastructure.

A final important aspect is that the actual sensor function virtualization should be transparent to end users. This means that any virtual functions that are added to devices should build on top of existing communication interfaces and that changes to protocols running on end hosts should be minimal and preferably non-existent. When SFV enhances physical devices, it should appear as if the constrained device offers the virtualized functions by itself from the user's point of view. When working with entirely virtualized devices, the interface to the user should be the same as the one that is used to communicate with constrained devices. If not, it will be cumbersome for users to discover and use the additional functionality.

Keeping in mind the requirements of the previous paragraphs, our proposed architecture is presented in figure 4. The basic principle is that sensor function virtualization is realized by decomposing the desired functionality into smaller modules. In the figure the modules are categorized according to the functionality that they provide (e.g. a module implementing 6LoWPAN compression would fall in the network category). The main benefit of this modular approach is that modules can be added at runtime (much like a plugin-based system) and that it allows to deploy modules over multiple machines thus improving scalability. The input and output data types for all types of modules are network packets. When network traffic arrives at a machine that provides sensor function virtualization, network packets flow up the architecture through a set of modules and down again through (possibly another set of modules) towards an outgoing network interface. The matching component takes care of passing network packets through the correct chain of modules based on configuration information that it stores. When combining all of these small modular

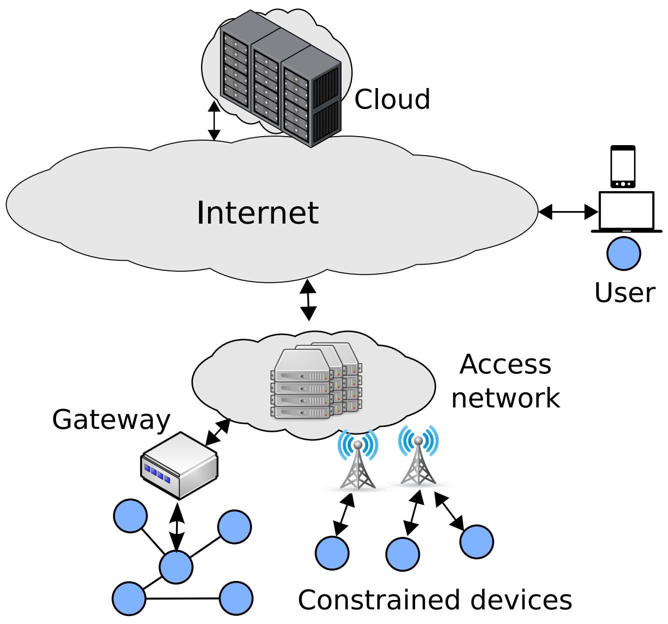


Fig. 5 Infrastructure at different locations throughout the Internet works together to provide sensor function virtualization in the Internet of Things

functional blocks that are running on a number of machines, the distributed processing and in turn sensor function virtualization for the IoT are realized.

Depending on the functionality that is to be virtualized, one or more locations in the network are suitable for deploying the functionality. For example, if local operation (i.e. within the same network) is required in case of Internet failure, virtualizing functionality on cloud infrastructure probably is unfeasible. In this case the local gateway will play an important role in sensor function virtualization. If on the other hand, global operation is required at all times then always-online systems (such as the cloud, but potentially also the gateway if it is available) can bridge situations where a constrained device is unavailable for communication (e.g. it might be sleeping to conserve energy, or its GPRS connection might be unavailable). Packaging SFV in modular components that can be (re)deployed at runtime, ensure that our architecture is able to handle this kind of flexibility.

Figure 5 gives an overview of the different locations where sensor function virtualization can be realized. The colored disks at the bottom represent constrained devices. Note that mobile device might migrate to a different access network in case of inter-network mobility (only one access network is shown in the figure). The figure also displays potential users on the right hand side (note that a constrained device is also considered as a user). As the Internet - and by extension most standardization efforts by IETF (see section 4) - follows a host-centric approach our SFV architecture is designed to be compatible with this point of view. While SFV in effect moves functionality away from Internet

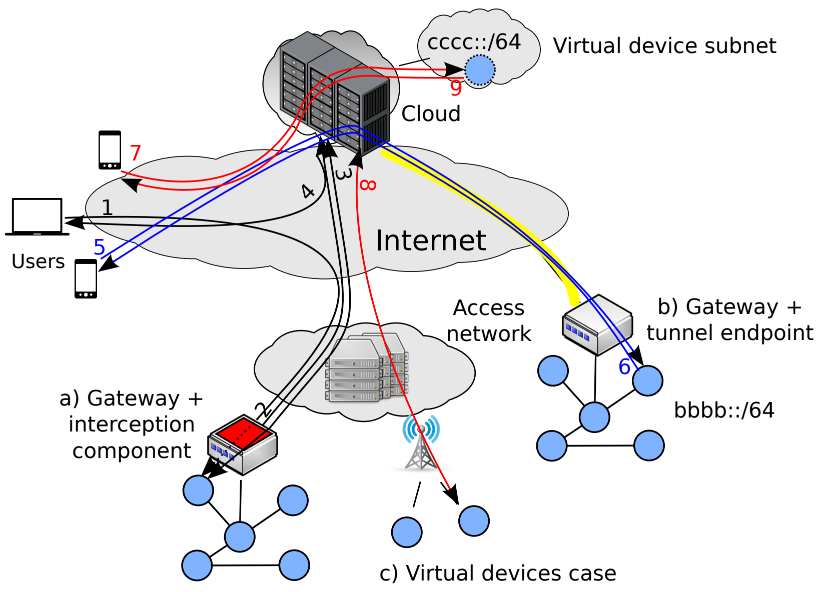


Fig. 6 Three different integration strategies for cloud-based SFV

hosts to supporting infrastructure, our approach allows extending constrained devices in a way that is transparent to its users. Transparency here means that it appears as if the functionality resides on the device itself, while in fact it is offered by supporting infrastructure. This is an important point, as this transparency differentiates our work from most of the cloud-based systems available today that each offer their own integration interface. This transparency means that users do not have to integrate with yet another cloud-based API as SFV functionality appears to be offered by the device itself. One consequence is that in order for a component of our architecture to enhance a constrained device with new functionality it must be able to receive and process requests for this device. For gateways that are on the routing path between a device and its Internet client this is trivial. For cloud-based infrastructure, extra measures are most likely necessary in all cases.

Figure 6 gives an overview of the mechanisms that ensure that cloud-based SFV infrastructure - or more generally any infrastructure that is not on the routing path between the user and the constrained device - is able to process requests destined and responses coming from the constrained devices that it handles. This is necessary if the cloud infrastructure should be able to modify requests and responses in order to fulfill its function in a transparent way (e.g. remove unnecessary data, add semantic descriptions, translate between data formats, etc.).

In case a (black arrows), an interception component deployed along the routing path is responsible for forwarding (matching) incoming network traffic for the constrained device (1) towards the cloud infrastructure (2). Once the request arrives in the cloud it can be processed, and a response can be generated immediately or a (modified) request can be sent towards the constrained device (3). In the latter case, the interception component is also responsible for ensuring that responses pass via the cloud. Once the cloud has processed the response (e.g. to update a cache), it forwards the response to the user (4). Note that response processing by the cloud is optional in certain cases (e.g. in the case of a cache that is only running on the gateway). Depending on the use case, the interception component can be configured by the cloud to stop forwarding traffic of a particular stream after the cloud has processed a number of packets of said stream (e.g. in access control the denial/granting of access is configured into the gateway by the cloud). This avoids unnecessary forwarding of traffic to the cloud. In this case the gateway can also forward responses directly to the user.

In case b (blue arrows), the constrained devices receive an IPv6 address that is globally routable and that is routed via the cloud infrastructure. As a result the cloud is able to process networking traffic destined to constrained devices (5). When the cloud forwards the traffic to the constrained device, then this traffic has to be encapsulated on the public Internet (otherwise it would never reach its destination). The tunnel endpoint can be a gateway (if one is available) or could be the constrained device itself (depending on its processing capabilities). The former is more suitable for WSN-like networks, while the latter is applicable to mobile nodes that are capable of hosting a tunnel endpoint. Note that the tunnel endpoint should also send all returning traffic through the tunnel in this case (6). If not, the cloud infrastructure is unable to process the responses from constrained devices (making e.g. caching impossible).

In case c (red arrows), the cloud-based SFV infrastructure mirrors every constrained device via a virtual device that is allocated a globally-routable IPv6 address from a “Virtual device subnet” that is routed to the cloud. In this case users interact with the virtual device in the cloud (7). Here the cloud fulfills the role of a traditional reverse proxy. This mode of operation is still transparent in the sense that the virtual device offers the same interfaces as the actual constrained devices but in this case the user does have to communicate with an Internet host at a different IPv6 address. The cloud infrastructure processes requests from users destined to virtual devices and forwards these to the actual constrained devices (8). The responses arrive from the constrained device and are sent back to user (9). This case is actually a generalization of case b. Note that this mechanism can also be used as a light-weight alternative to mobile IPv6 for constrained devices where their anchor point on the Internet changes due to their mobility. In this case, users always communicate with the fixed “virtual device” and the cloud takes care of mapping requests to the volatile IP endpoint of the constrained device. Signaling of the volatile IP address can happen via existing interfaces.

4 IETF protocol stack for the Internet of Things

As the remainder of this paper will employ CoAP and the IETF IoT stack for illustrating SFV, a concise overview of all protocols involved is presented here. Ishaq et. al present a more elaborate overview of the subject in [8].

The IETF started standardization for the IoT with an adaptation protocol for IPv6 over 802.15.4 communication links. The resulting protocol is defined by RFCs 4944 [11] and 6282 [7] and is more generally known as 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks). 6LoWPAN allows compressing IPv6 and UDP packets by eliding redundant information in IPv6 and UDP headers. 6LoWPAN also provides fragmentation support for large IPv6 packets, thus fulfilling the minimum MTU requirement of 1280 bytes found in IPv6. As a result, deploying IPv6 (with its 40 bytes header) on links with small MTUs (such as 802.15.4 links with a 127B MTU) in a standard compliant matter has been possible since 2007. It is interesting to note that the idea of compression for IPv6 in 6LoWPAN is also being applied to other protocols (such as DTLS [13]).

Around 2008 consensus emerged in the IETF on standardizing a routing protocol for use in low power and lossy networks (LLNs). Due to the unique properties of LLNs (e.g. conserve energy as much as possible, point to multipoint traffic, etc.), their routing requirements differ from what traditional routing protocols considered at that time. The results of the ROLL working group include a number of requirements for specific types of LLNs (home automation, industrial, smart city, etc.) and a routing protocol known as the "IPv6 Routing Protocol for Low-Power and Lossy Networks" (RPL). RPL, standardized in RFC 6550 [21], allows constructing routing trees (known as DODAGs) for LLNs according to an objective function that can be tuned to meet the requirements specific to the type of LLN.

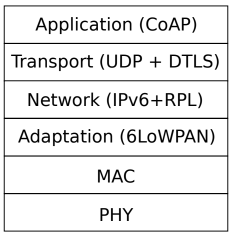


Fig. 7 IETF protocol stack for low power and lossy networks in the Internet of Things

In June 2014 the IETF Constrained RESTful Enviroments (core) working group standardized an application layer protocol for use in low power and lossy networks. The Constrained Application Protocol (CoAP), as defined in RFC 7252 [16], allows building applications based on the concepts of RESTful web services that are well-known from the WWW. CoAP can be thought of as a lightweight alternative to HTTP and as the counterpart of HTTP for use in the embedded world (with battery operated devices, unreliable wireless links and low-cost 16 bit microcontrollers). Another important contribution of the working group is RFC 6690 [15] which specifies a format for web linking for constrained web servers. This format can describe hosted resources, provide attributes for resources and define relationships between links. It is designed with a simple parser in mind that is memory efficient and that provides compact web links. Figure 8 shows a typical request/response message exchange between a CoAP client and server. First the client discovers which resources are hosted by the server via the ".well-known/core" resource, the response contains a list of links according to the core link format. The second request retrieves a plain-text representation of the temperature resource from the server.

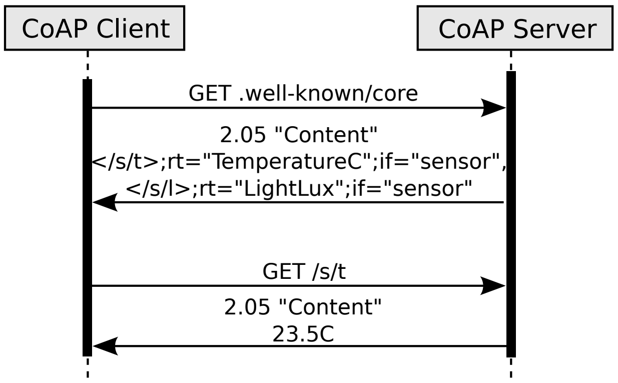


Fig. 8 A CoAP request/response message exchange showing resource discovery and data retrieval

The CoRE working group identified that security is an important requirement for almost all IoT applications. Therefor it chose to rely on transport-layer security as this is a popular choice in the WWW today. More specifically, RFC 7252 defines a binding for CoAP that specifies how CoAP should be used in conjunction with DTLS. Datagram Transport Layer Security (DTLS, RFC 6347 [14]) provides end-to-end security via TLS over UDP (as opposed to TCP in case of TLS). When employing pre-shared key (PSK) cipher suites, DTLS can rely solely on symmetric encryption for providing end-to-end security. When combined with the small memory footprint of DTLS, this makes DTLS an attractive candidate for use in LLNs.

Finally, the IETF has identified the need for standardizing a set of DTLS parameters for use in LLNs. To this end, the DTLS In Constrained Environments (DICE) working group was formed in 2013. DICE is looking to standardize a LLN profile for DTLS, to overcome some of the issues of DTLS in combination with multicast and to investigate practical issues surrounding the DTLS handshake in LLNs. DICE explicitly states that it will not alter the DTLS state machine.

Note that for GPRS equipped constrained devices, the 6LoWPAN and RPL standards are less applicable. These types of device do however still benefit from the low communication overhead offered by DTLS/CoAP when compared to TLS/HTTP [5]. An overview of the entire stack is shown in figure 7.

5 SFV in the unconstrained domain

As mentioned function virtualization enables us to extend constrained devices with new functions without burdening the constrained device itself. This is achieved by offloading (or virtualizing) functionality to more powerful infrastructure (such as gateways, routers, cloud-based infrastructure) in a way that is transparent to the constrained device and its users. This section presents two scenarios that demonstrate SFV according to the architecture presented in section 3. Note that these two scenarios serve as illustrations of SFV and that SFV as a technique is not limited to what is shown here (for example SFV at the network layer is not shown here). The scenarios are applied to the IETF IoT stack that was detailed in the previous section.

The first case considers constrained devices that are only intermittently reachable via the Internet. These devices usually sleep for prolonged periods of time in order to keep energy consumption to a minimum. Every once in a while (e.g. after a certain period of time has passed or after an event is triggered) the device wakes up, sends its sensor data to a so-called mirror server, checks whether there is any incoming (configuration) data and goes back to sleep. Users do not interact with the sleepy device, instead they communicate with a mirror server that is always online. In CoAP this kind of functionality is known as a CoRE Mirror Server [20]. One problem with this approach is that mirror servers host mirrored resources on behalf of a multitude of sleepy devices. As a result, the user is exposed to the mirrored resources belonging to all sleepy devices that are interacting with the mirror server. This is problematic when the user in question is unfamiliar with and does not support the mirror server client operation interface. As a solution, SFV hides this client operation interface and instead provides a dedicated CoAP server for every device that is mirrored on the mirror server. This dedicated CoAP endpoint is always online and hides the underlying mirror server's interfaces from CoAP clients.

Apart from hiding the mirror server's client operation interface to clients, this scenario also illustrates how SFV can extend a device by hosting new CoAP resources on behalf of the device. In this case, SFV emulates CoAP resources via the dedicated CoAP endpoint that provides semantic descriptions of the resources offered by the constrained device. Because these descriptions are too large to store on the device itself, it is more efficient to offer them by means of SFV.

In order to realize this scenario in accordance with the architecture presented in figure 4, two SFV modules are needed. The MS module provides the dedicated CoAP endpoint that hides the mirror server's interface. The EMU module implements the CoAP resource emulation. The matching component

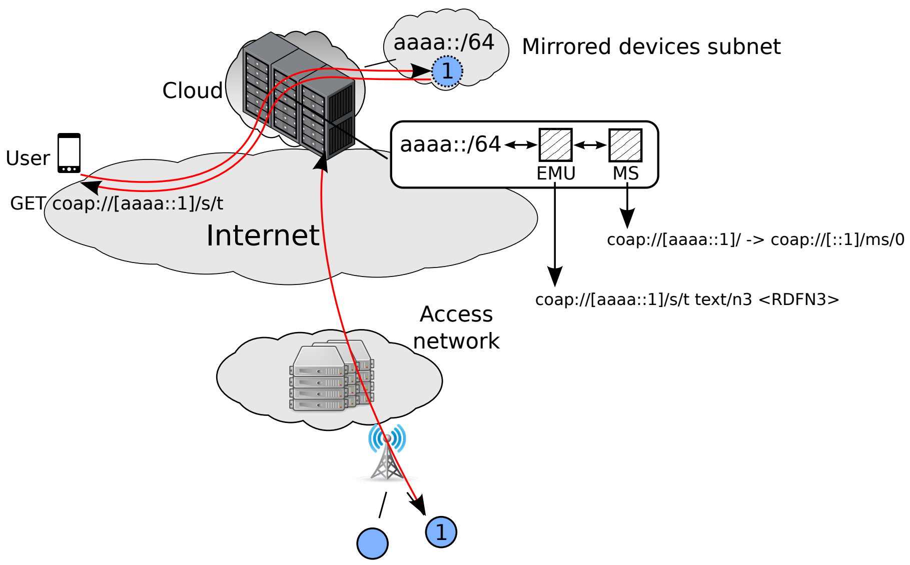


Fig. 9 Two SFV modules providing emulated resources (EMU) and a mirror server abstraction (MS)

also has to be configured in order to match requests for CoAP resources on the dedicated endpoint to the two modules. The result is shown in figure 9. The constrained device (near the bottom in the figure) wakes up periodically to push its sensor data to the mirror server in the cloud as per [20]. Users send their CoAP requests to the mirrored devices, which reside in a subnet that is routed towards the cloud (aaaa::/64 in the figure). Requests are handled by the SFV matching component in the cloud (the white box in the figure). The matching component passes the requests to the EMU module. If this module does not generate a response, then the request is passed along to the MS module. In the figure, the EMU module is configured to emulate one resource at coap://[aaaa::1]/s/t for the content-type "text/n3". This emulated resource provides a semantic description of a temperature sensor in the RDF Notation3 format. The response of the resource (<RDFN3>) is shown in the listing below.

```
<coap://[aaaa::1]/s/t> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
<http://purl.oclc.org/NET/ssnx/ssn#Sensor>;
<http://spitfire-project.eu/ontology/ns/obs> <http://vmuss07.deri.ie:8182/ld4s/res/property/temperature>;
<http://spitfire-project.eu/ontology/ns/uom> <http://vmuss07.deri.ie:8182/ld4s/res/uom/centigrade>;
<http://purl.oclc.org/NET/ssnx/ssn#onPlatform> <coap://[aaaa::1]>;
<http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#hasLocation>
"Testbed iMinds;
<http://purl.oclc.org/NET/ssnx/ssn#featureOfInterest> "iMinds Office"
.
```

Listing 1: RDFN3 description for a temperature CoAP resource

If the EMU module does not generate a response, then the request is handled by the MS module. The MS module simply rewrites CoAP requests for mirrored devices to requests for the mirror server. To accomplish this, it stores a mapping between the two. For example, in the figure a request to coap://[aaaa::1]:/s/t will be mapped to coap://[::1]/ms/0/s/t. The user is totally oblivious that its request is rewritten by the module. Responses sent to the user by the MS module use the destination address of the original request as the source address, e.g. for the example responses come from aaaa::1.

The second case considers a Wireless Sensor and Actuator Network (WSAN) that deploys the full IETF IoT stack from figure 7 and that relies on DTLS for end-to-end security. In this network all constrained devices are accessible via their publicly routable IPv6 address and users interact with the CoAP resources that are hosted on the constrained devices. Optionally, devices employ radio duty cycling to prolong battery life. In such a scenario, DTLS sessions with the sensors and actuators are typically restricted to cipher suites that exclude any asymmetrical encryption due to the high complexity and computational cost of the algorithms involved. Furthermore, transporting and verifying certificates as is necessary in a Public Key Infrastructure (PKI) is also deemed too expensive for sensors and actuators. Instead, the constrained devices rely on symmetrical encryption algorithms (AES being a common choice) and Pre-Shared Key (PSK) cipher suites (TLS_PSK_WITH_AES_128_CCM_8 is a popular suite) due to the relatively simple algorithms (AES co-processors are common) and small amount of keying material that has to be communicated respectively.

One issue in this scenario is that PSK cipher suites do not scale well as the number of clients increases. A sensor and or actuator (sensa) has to share a unique pre-shared key with every client. Obviously, this does not scale. A second issue arises due to the end-to-end encryption (E2EE) itself. As one would expect in E2EE, intermediary systems are unaware of the contents of the requests/responses between a client and the sensa. However, in WSANs processing at the edge of the network by a trusted intermediary can significantly improve response times and battery lifetimes. This consideration applies to techniques such as caching and access control. Ideally, we would want a variant of E2E security where the WSAN gateway (which is considered to be trusted by both the client and the sensa) is able to decipher communications in order to provide caching, access control, etc. To summarize, the low scalability of PSK cipher suites and the inability to perform any processing at the edge of the network (e.g. caching) are two problems inherent to DTLS in WSANs today.

SFV can provide a solution to these two problems by virtualizing asymmetrical encryption and by extending the WSAN gateway up to the application layer (in effect realizing an application-level gateway). SFV on the trusted gateway intercepts traffic and terminates DTLS sessions between clients and sensas. As a result, the DTLS handshake of the client is performed with the trusted gateway. Note that this happens completely transparent to the client, i.e. the client still communicates with the public IPv6 address of the sensa (as

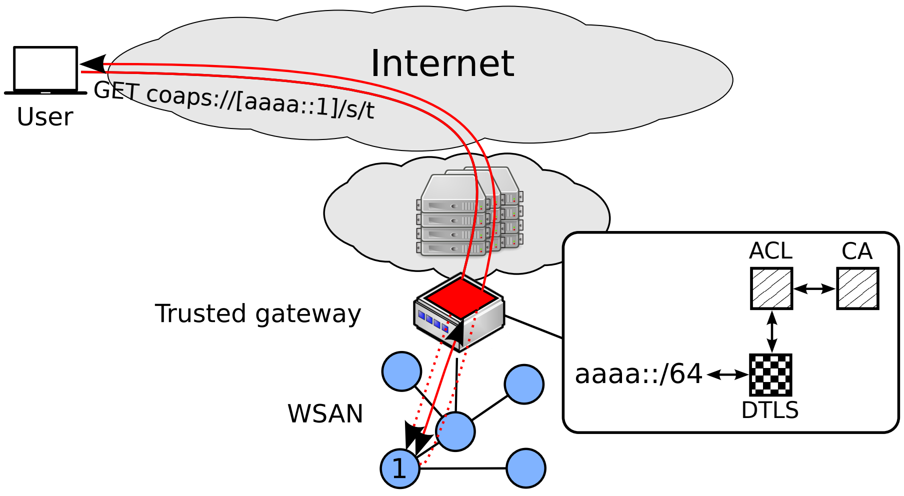


Fig. 10 SFV at a trusted gateway provides DTLS termination, access control and caching

the gateway is assumed to be trusted, this is considered secure). The trusted gateway performs a DTLS handshake with the sensa using a PSK cipher suite and all DTLS traffic of the client is sent over this session. This way, the sensa only has to share a unique PSK with the gateway. Furthermore, the gateway can offer a more extensive list of supported cipher suites to the client. This list can include suites that rely on PKI, thereby circumventing the scalability problem of the PSK suite offered by sensas. Finally, by terminating DTLS sessions the trusted gateway is also able to perform edge processing such as caching and access control.

Figure 10 presents an overview of this case. Preliminary results show that terminating DTLS and re-using the same DTLS session in the WSAN for multiple clients can reduce the energy consumption of the constrained device by more than 59% when compared to traditional end-to-end security where the trusted gateway does not terminate the session. This large reduction in expended energy is primarily due to the fact that the constrained device only has to setup one DTLS session with the trusted gateway for all clients. As a result, the expensive DTLS handshake is only executed once whereas in the traditional end-to-end case the handshake has to be repeated for every separate client. Because the gateway is able to see the contents of the CoAP requests, it can apply access control on a per resource basis. For example, users that are authenticated with a specific certificate might be able to reconfigure resources (i.e. PUT and POST methods are allowed for specific resources), whereas others are only allowed to retrieve data (i.e. only the GET method is allowed for specific resources). Furthermore, the DTLS termination module can be configured to omit DTLS in the WSAN for certain requests that do not require authentication and/or confidentiality in the WSAN. For the latter, consider an example where retrieving a temperature value is done in plain-text in the WSAN, but is transferred in cipher text over the public Internet. Note that these kinds of advanced scenarios are very hard to realize without a distributed approach due to constraints of the IoT devices. This is where SFV really shows its potential.

6 SFV in the constrained domain

In some cases, sensor function virtualization might be required on the constrained devices themselves. For SFV on constrained devices the architecture from figure 6 is often not a good fit, as their characteristics differ greatly from what is available on more powerful hardware such gateways and the cloud. Nevertheless, in previous work we have shown that some form of flexible distributed computing can be provided for constrained devices. Other approaches that target reprogrammability of constrained devices can also achieve the desired flexibility at a higher cost due to the expensive operation of transferring a program's memory contents.

Bindings [17] allow a third party to setup direct CoAP-based interactions between sensors and actuators and other devices. One of the benefits is that the third party does not have to be a message broker between the sensor and the actuator and therefor can go offline after setting up the binding. Furthermore, this gives us the flexibility to link sensors to actuators after they are deployed instead of when they are produced.

Another concept that we have presented in previous work [18] are small REST web services that fulfill common tasks such as filtering and (de)multiplexing of sensor data. These web services can be deployed inside the constrained network (on the gateway but also on constrained devices) and are called RESTlets. RESTlets have one or more inputs and zero or more outputs. All in- and outputs are RESTful web services. RESTlets allow decomposing parts of the data processing into smaller blocks that can be deployed close to the data sources and their consumers. Because requests and responses do not have to traverse the entire WSAN, response times and energy consumption are lowered in multi-hop WSANs.

7 Related work

Thirty-five years after Cerf V. and Kirstein P. argued about the role of gateways in what was then the early Internet, the authors revisit the same question at the advent of the Internet of Things [4]. Very similar to the early Internet, there exist a number of proprietary protocols for connecting things to the Internet today. The authors argue that there will first be a period of adaptation for connecting legacy technologies to the IoT, followed by period of adoption where the IP will supersede these legacy technologies. In terms of adaptation, the authors introduce repositories as the end-point for interactions of clients with things. These repositories have direct access to sensors to gather data or to set configuration information. Repositories act as a bridge between clients and sensors and apply adaptations where necessary. The authors also mention that repositories can be part of a distributed system. We envision that these repositories are deployment options for our SFV concept. Our work does not only focus on adapting proprietary technologies for Internet integration, SFV also enhances things that already deploy the IETF IP stack. This is necessary in order to bridge the gap in processing power and capabilities between constrained devices and traditional Internet devices.

Varakliotis et al. [19] build further on the vision presented by Cerf V. & Kirstein P. The authors describe which tasks are common for a gateway and decompose these into smaller blocks that are situated at various layers in the protocol stack. These functional blocks are very often required but need not be co-located on one and the same machine, e.g. they could be distributed on multiple servers. The result is a distributed protocol stack, with a minimal controller and external servers that hold most of the functional blocks. The IETF stack is also considered as one of the potential technologies for IoT networks (i.e. the DevNet). Our vision aligns closely to that of Varakliotis et al. However, the assumption that a distributed approach will automatically lead to a stripped gateway with minimal functionality and external servers doing the heavy-lifting does not hold in all cases. Consider an example where things generate large amounts of data that cannot be transported in their raw format to external infrastructure. In such a case, it can be beneficial to do some form of processing locally at the gateway. In contrast to the work of Varakliotis et al., our SFV architecture does not exclude this possibility.

Another interesting work to mention in this content is the IPv6 addressing proxy by Jara et al. In [9] an adaptation proxy is presented for mapping native addressing from legacy technologies to the IPv6 Internet of Things. The authors provide extensive examples for legacy technologies such as X10, EIB, CAN and RFID from the industrial, home automatic and logistics domains. Our SFV concept is an ideal candidate for implementing such an addressing proxy.

In "Moving application logic from the firmware to the cloud" [10] Kovatsch et al. argue that application development should be fully decoupled from the embedded domain. Embedded devices are thin servers that export only elementary functionality by means of REST resources. The approach relies on application servers that house the logic of applications. Our SFV concept can be seen as a distributed version of these application servers. SFV also allows to transparently enhance embedded devices for all clients involved, thus the virtualized functionality becomes available to all parties interacting with the device and not only to the application server. As a result, our work should facilitate easier reuse of application logic than the application servers presented by Kovatsch et al.

PyoT [3] is a programming framework for the IoT that claims to simplify IoT application development by providing a number of common operations to developers. Supported operations include discovery, monitoring, storage and triggering of devices and their data. The PyoT framework can reside on a gateway or in the cloud. Furthermore, PyoT also integrates with T-Res [2], a framework enabling “in-network processing” in IoT-based WSNs. The existence of the PyoT framework shows that extending constrained devices with common operations can help to accommodate IoT application development. This is also one of the objectives of SFV. T-Res is an alternative approach to SFV in the constrained domain, as discussed in section 6, that relies on programmability via a minimal python interpreter suitable for embedded systems (PyMite).

Finally, there are two other works that discuss the role of cloud computing in the Internet of Things. In [12] Pereira et al. present a holistic network architecture for supporting mobility in IoT applications. This service-oriented architecture can be deployed on three levels: on the node itself (for resilience against lost Internet connectivity), on a SOA-enabled gateway (when more processing than what a node can provide is necessary) and on the Internet (where all limitations of processing power are mitigated). In [22], Zhou et al. present a common architecture for integrating the Internet of Things with cloud computing that is named CloudThings. The authors describe a cloud-based Internet of Things platform for developing, deploying, running, and composing Things applications. CloudThings differs from PyoT in that it focuses on providing a development platform (that supports IaaS, PaaS but also SaaS) as opposed to a more rigid development framework. These two works illustrate how distributed processing can extend constrained IoT devices, which is one of the key concepts of SFV.

None of the identified related work considers reconfiguring the network to optimally support an IoT application’s requirements as an integral part of their approach. In our work this is an important aspect of distributed intelligence, as in some cases (cfr. section 2.2.1) the network has to be able to adapt in order to fulfill the requirements of the IoT application.

8 Conclusions

In this paper we presented some of the limitations that are present in today’s IoT systems. We argued that some of these issues can be overcome by means of distributed intelligence. In distributed intelligence the network and its infrastructure can be reconfigured to meet application requirements and the resources offered by these systems can be leveraged to provide processing and communication at the right place and time according to the requirements of IoT applications. Sensor function virtualization is a technique that enables distributed intelligence via modular functional blocks that can be deployed anywhere in the network infrastructure. This way processing can be placed where it is most needed. Furthermore, SFV can also be used to expose configuration interfaces for existing network functionality (thus enabling the reconfiguration required in distributed intelligence). By means of two extensive examples, we have illustrated some of the benefits that SFV can bring to the table of IoT application development and distributed IoT systems.

In the future, we plan to evaluate our approach more thoroughly. The reduction in energy consumption and latency in the DTLS termination use case already show that sensor function virtualization can have a significant impact on important performance metrics for constrained devices and IoT applications. However, more experiments are necessary. Furthermore, reconfiguring the communication network (e.g. routing, MAC protocols, etc.) in order to support specific communication patterns required by an IoT application was only briefly discussed. More work is needed to explore what is possible and how well this aligns with the vision of distributed intelligence that was outlined in this paper.

Acknowledgement

The authors would like to acknowledge that part of this research was supported by the COMACOD project. The iMinds COMACOD project is co-funded by iMinds (Interdisciplinary institute for Technology) a research institute founded by the Flemish Government. Partners involved in the project are Multicap, oneAccess, Track4C, Invenso and Trimble, with project support of IWT.

References

1. Light-Weight Implementation Guidance (lwig) - Charter (2011). URL `https://datatracker.ietf.org/wg/lwig/charter/`
2. Alessandrelli, D., Petraccay, M., Pagano, P.: T-Res: Enabling Reconfigurable In-network Processing in IoT-based WSNs. In: 2013 IEEE International Conference on Distributed Computing in Sensor Systems, pp. 337–344. IEEE (2013). DOI 10.1109/DCOSS.2013.75. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6569453`
3. Azzara, A., Alessandrelli, D., Bocchino, S., Petracca, M., Pagano, P.: PyoT, a macroprogramming framework for the Internet of Things. Proceedings of the 9th IEEE International Symposium on Industrial Embedded Systems (SIES 2014) (i), 96–103 (2014). DOI 10.1109/SIES.2014.6871193. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6871193`
4. Cerf, V., Kirstein, P.: Gateways for the Internet of Things-An Old Problem Revisited pp. 2641–2647 (2013). URL `http://discovery.ucl.ac.uk/1414930/`
5. Daniel, L., Kojo, M., Latvala, M.: Experimental Evaluation of the CoAP, HTTP and SPDY Transport Services for Internet of Things. In: 7th International Conference on Internet and Distributed Computing Systems, pp. 111–123 (2014)
6. Evans, D.: The internet of things: how the next evolution of the internet is changing everything. CISCO white paper (2011). URL `http://scholar.google.be/scholar?cluster=12792589873848912975&hl=nl&as_sdt=2005&sciodt=0,5#0`
7. Hui, J., Thubert, P.: RFC 6282: Compression format for IPv6 datagrams over IEEE 802.15. 4-based networks (2011). URL `https://tools.ietf.org/html/rfc6282.txt`
8. Ishaq, I., Carels, D., Teklemariam, G.K., Hoebeke, J., Van den Abeele, F., De Poorter, E., Moerman, I., Demeester, P.: IETF standardization in the field of the Internet of Things (IoT): a survey. Journal of Sensor and Actuator Networks **2**(2), 235–287 (2013)

9. Jara, A.J., Moreno-Sanchez, P., Skarmeta, A.F., Varakliotis, S., Kirstein, P.: IPv6 addressing proxy: mapping native addressing from legacy technologies and devices to the Internet of Things (IPv6). Sensors (Basel, Switzerland) **13**(5), 6687–712 (2013). DOI 10.3390/s130506687. URL `http://www.mdpi.com/1424-8220/13/5/6687/htm`
10. Kovatsch, M., Mayer, S., Ostermaier, B.: Moving application logic from the firmware to the cloud: Towards the thin server architecture for the internet of things. . . . Mobile and Internet . . . (2012). URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6296948`
11. Montenegro, G., Kushalnagar, N., Hui, J., Culler, D.: RFC 4944: Transmission of IPv6 packets over IEEE 802.15. 4 networks (2007). URL `https://tools.ietf.org/html/rfc4944`
12. Pereira, P.P., Eliasson, J., Kyusakov, R., Delsing, J., Raayatinezhad, A., Johansson, M.: Enabling Cloud Connectivity for Mobile Internet of Things Applications. 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering pp. 518–526 (2013). DOI 10.1109/SOSE.2013.33. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6525570`
13. Raza, S., Shafagh, H., Hewage, K., Hummen, R., Voigt, T.: Lithe: Lightweight secure CoAP for the internet of things. IEEE Sensors Journal **13**, 3711–3720 (2013). DOI 10.1109/JSEN.2013.2277656
14. Rescorla, E., Modadugu, N.: RFC 6347: Datagram transport layer security version 1.2 (2012). URL `https://tools.ietf.org/html/rfc6347`
15. Shelby, Z.: RFC 6690: Constrained RESTful Environments (CoRE) Link Format (2012). URL `https://tools.ietf.org/html/rfc6690`
16. Shelby, Z., Hartke, K., Bormann, C., Frank, B.: RFC 7252: Constrained Application Protocol (CoAP) (2014). URL `https://tools.ietf.org/html/rfc7252`
17. Teklemariam, G.K., Hoebeke, J., Moerman, I., Demeester, P.: Flexible, direct interactions between CoAP-enabled IoT devices. In: The Eighth International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS-2014) (2014)
18. Teklemariam, G.K., Hoebeke, J., Van den Abeele, F., Moerman, I., Demeester, P.: Simple RESTful Sensor Application Development Model Using CoAP. In: 9th IEEE Workshop on Practical Issues in Building Sensor Network Applications (IEEE SenseApp 2014), pp. 552–556 (2014)
19. Varakliotis, S., Kirstein, P.T., Jara, A., Skarmeta, A.: A process-based Internet of Things. In: 2014 IEEE World Forum on Internet of Things (WF-IoT), pp. 73–78. Ieee (2014). DOI 10.1109/WF-IoT.2014.6803123. URL `http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6803123`
20. Vial, M.: CoRE Mirror Server (draft-vial-core-mirror-server-01) (2013). URL `http://tools.ietf.org/html/vial-core-mirror-server-01`
21. Winter, T., Thubert, P.: RFC 6550: RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks (2012). URL `https://tools.ietf.org/html/rfc6550`
22. Zhou, J., Leppanen, T., Harjula, E., Ylianttila, M., Ojala, T., Yu, C., Jin, H.: CloudThings: A common architecture for integrating the Internet of Things with Cloud Computing. In: Proceedings of the 2013 IEEE 17th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2013, pp. 651–657 (2013). DOI 10.1109/CSCWD.2013.6581037