Refactoring Internet of Things middleware through Software-Defined Network

Lucas M. R. Arbiza, Leandro M. Bertholdo, Carlos Raniery P. dos Santos,
Lisandro Z. Granville, Liane M. R. Tarouco
Informatics Institute
Federal University of Rio Grande do Sul
Porto Alegre, Brazil
{lmrarbiza, berthold, crpsantos, granville, liane}@inf.ufrgs.br

ABSTRACT

Refactoring means to rewrite pieces of code aiming to improve it internally but keeping the expected software behavior. In this paper we present the refactoring of an Internet of Things middleware based on Software-Defined Network. In a previous work we proposed a middleware to address issues we found in healthcare devices used to monitor patients with chronic illnesses in their homes. Software-Defined Network allowed the redesign of the middleware architecture to improve *things* management, its interconnection with services, and the deployment process of new monitoring scenarios. Refactoring process also extended the middleware to support multiple services in a single home network sharing the same network infrastructure. This work details an OpenFlow controller and an application developed to achieve our goals; we also present sample scenarios where our approach can be applied showing different services delivered in the same home network environment, and using data from all connected devices to build a digital representation of the physical realm.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Internet of Things middleware; C.2.3 [**Network Operations**]: Network management; C.2.4 [**Distributed Systems**]: Network operating system—*Software-Defined Network*

General Terms

Design, Experimentation, Management

Keywords

Software-Defined Network, Internet of Things, Middleware, Refactoring

SAC'15 April 13-17, 2015, Salamanca, Spain.

http://dx.doi.org/10.1145/2695664.2695861

1. INTRODUCTION

Modern communication networks are demanding the use of new and sophisticated management solutions. While such networks used to be composed of traditional devices (*e.g.,* routers, servers, PCs), today, the number and variety of connected elements has increased. Such elements, usually referenced to as *things*, can vary in many aspects, such as size, computing resources, operating mode, network technologies and protocols. This scenario is known as Internet of Things (IoT), a term originally defined in 1999 to describe the RFID tags used to track objects in supply chains. Nowadays, the term means the digital representation of physical environment, created using data sent by multiple sensing devices [14].

The importance of research in IoT is evidenced by the growth in the number of heterogeneous devices connected to the Internet — the industry predicts about 15 billions by 2015 and 50 billions by 2020 [7]. These devices are used for many and diverse applications, for example healthcare, home automation, fitness tracking, environment monitoring, and in smart cities. Integrating heterogeneous devices into existing network infrastructure is a complex task. In a previous effort to develop IoT services for smart cities [5], we have identified a set of important aspects to be considered when employing IoT devices as part of a service. These aspects are presented in the following:

- Interconnection: most of the Internet-enabled devices (*i.e.*, *things*) available today employ proprietary communication methods to connect with the manufacturer servers or with other devices from the same manufacturer. Such operation mode is know as *silo* [1,11,13,17]. The main problem with *silos* is that data are closed in a proprietary solution; even when accessible through APIs (Application Programming Interface), it is first sent to proprietary servers;
- Security and privacy: data collected by the devices are usually transmitted in plain text and there is no authentication between device and destination server;
- Management: IoT devices usually employ sleep schedule mechanisms to reduce the power consumption, avoiding traditional network management solutions (*e.g.,* SNMP, NETCONF, ICMP-based connectivity monitoring) to be employed;
- Data structure: according to [12], 80% of healthcare data is unstructured, thus requiring data-mining in

large volumes of data in order to find relevant information to foresee future outcomes in patients health.

In order to tackle such issues, and considering the working scenario of REMOA project (health monitoring of patients with chronic illnesses), we developed a middleware to execute on Access Points (APs) [16]. According to [9], middleware is one or more software layers employed to abstract technological details to the application level. The middleware development allowed us to identify that resource constraints of target environment (APs) slow down development of required modules, consequently making the monitoring solution hard to scale, deploy, and improve.

In this work we refactor the REMOA middleware in order to support Software-Defined Network (SDN) technologies as underlying network substrate. Specifically, we refactored two important features provided by the middleware: *things* management and interconnection. Inspired by [10], the middleware functions implemented to run in APs were moved to remote servers, except the switching and routing tasks, now performed as an OpenFlow switch. In the refactoring process we also extended middleware to support more than one service in the same AP.

In SDN-based networks, forwarding tables of switches are configured by controllers, which are entities responsible for storing and distributing flow rules and actions. Once an unmatched packet arrives at a switch, it contacts the network controller, which can decide to modify the current flow-table rules, or to deploy new rules. This approach presents advantages for IoT environments since the APs can be updated by the SDN controller once new monitoring devices are added to the network and start sending packets.

In our refactored middleware, services run on remote servers and APs forward network packets based on flow rules received from the SDN controller. Since the servers do not present the same hardware limitations as the APs have, more sophisticated services (*e.g.*, management tasks, data receiving and processing) can be created and deployed to the IoT environment in a fast and dynamic fashion; particularly because developers can make use of development resources not possible to be used when all the middleware was being developed targeting APs. Another benefit enabled by employing SDN in IoT context is the view of the networked environment as a whole; services may use information about communication of every connected device, even *things* belonging to other services, to expand the digital representation of the environment beyond that provided by data from their devices.

This paper is structured as follows: In section 2 we provide an overview of the SDN paradigm and present other efforts where the SDN paradigm is investigated in the IoT scenario. Section 3 depicts the REMOA middleware, its components, and development decisions. Middleware refactoring is detailed in section 4 where we present the architecture of our approach and its components: Derailleur OpenFlow controller and ThingsFlow application (sections 4.1 and 4.2). In section 5 we provide some examples of services and scenarios where this approach can be deployed. Finally we present our considerations and point out next steps in our work.

2. RELATED WORK

SDN is a network paradigm that has been largely explored in the last years, both in industry and academia. The main feature of SDN is decoupling the network control plane from the forwarding devices, allowing to define the network logic through applications that configure forwarding rules on SDN switches [6].

In IoT field, SDN started to be explored more recently. Qin *et al.* [15] propose an architecture where SDN is used to enable different services to cooperate. The authors exemplify their proposal in a road scenario where, for example: vehicles receive notifications about traffic and accidents; vehicles communicate with each other; travel planning of electric cars is optimized based on the location of recharging stations; cabs can be located. In that work, vehicles are recognized based in data themselves provide. Such data are crossed with data stored in databases; once a vehicle is recognized, its data can be sent to other services, such as a network of cameras, that will visually locate a vehicle (*e.g.*, cab) based on geo-location data.

The work of Chetty and Feamster [10] and Yiakoumis *et al.* [18] does not address IoT explicitly, but their efforts tackle home networks issues and use SDN to develop their refactoring proposals. Chetty and Feamster [10] succeeded in moving network complexity away from the home end-users. End-users often do not properly configure their private networks because of the specificity of technical information presented to them, such as protocols, packet types, attack types, and Maximum Transmission Unit (MTU). The authors propose that network configuration possibilities are presented in a high level abstraction configuration interface; SDN is used to translate from configuration interface to technical OpenFlow settings.

Yiakoumis *et al.* [18] propose to slice home network allowing different service providers (*e.g.,* broadband, electric energy, gas, NetFlix, GoogleTV) to use different slices to deliver their services sharing the same network infrastructure, aiming at costs reduction. Each service cannot interfere in other services so that slices isolation is an essential feature. Each slice is controlled by a service provider. For each slice, it is guaranteed isolation of traffic and broadband, independent control and the possibility to be changed and/or customized by the service provider.

The target scenario of this paper is also home networks, but with the characteristics of the monitoring environment of the REMOA project. Laying over the ideas of the previously cited work, we extended the REMOA middleware through a refactoring process, but we kept focus on IoT issues. In the following sections we describe how refactoring has been carried out, as well the redesigned architecture and the implemented components.

3. REMOA MIDDLEWARE

The REMOA middleware was developed to address common issues found in IoT environments, mainly focused on our health monitoring scenario: interconnection, security and privacy, management, and data structure [16]. The middleware was developed considering the TP-Link WR1043ND AP, with architecture presented in Figure 1, and the main components discussed in the following:

1. Transparent Proxy: this component is responsible for analyzing all the outgoing traffic to the Internet. It receives health data from *things*, sends to the healthcare application, and records in the MIB (Management

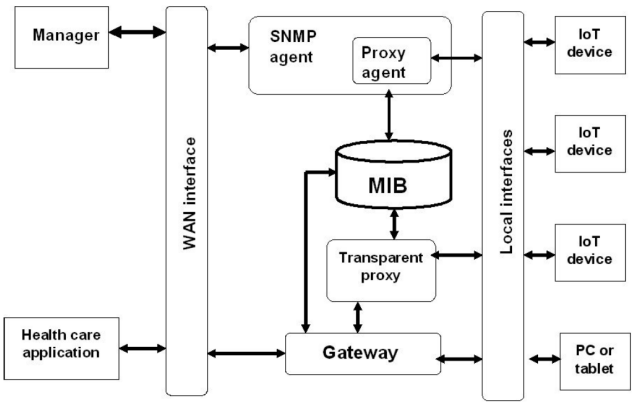

Figure 1: REMOA Middleware Architecture [16].

Information Base) information about *things* operation to be used for management purposes; when necessary it applies filtering or redirecting rules to the network traffic;

2. SNMP agent and Proxy agent: responsible for handling SNMP requests made to *things* with no SNMP support, it replies to requests with data previously stored in the MIB by Transparent Proxy module;

3. Gateway: enables transparent communication between the healthcare application and *things*, but only when *things* operating mode allows them to be accessed. When necessary, Gateway translates messages from one technology to another (*e.g.*, Ethernet to 802.15.4-based technology).

The middleware was designed to execute over the OpenWRT [4], a Linux-based operating system for embedded devices, which is widely employed in domestic APs. In order to reduce resources consumption of APs (*e.g.*, memory and processing), the middleware was developed using the C++ instead of other resources demanding programming languages, such as Java and Python.

With C++ there is a gain in performance and resources saving running native code, but developing in C++ use to be slower than developing with languages featured to abstract type conversions, pointers, memory allocation/deallocation, etc. C++11/C++14 features make development easier, but still demands lots of work, for example, to implement a Transparent Proxy module to communicate with *things* in a REST (Representational State Transfer) style.

The deployment of new middleware features, or to employ new monitoring devices in REMOA approach is a process that requires much working efforts, *i.e.*, the development of specific modules having as target a resource constrained system and architecture of the AP followed by the update process. If AP is already running in the house of a patient it is updated remotely; if some error occur the monitoring is compromised, in addition technical staff will need to move to the patient house for maintenance.

4. MIDDLEWARE REFACTORING

Figure 2 illustrates the middleware refactoring. The complexity of network services provided by modules earlier implemented over the AP, now is distributed among servers provided with more computing power. AP now acts as an OpenFlow switch and establishes IPSec connection with the servers to where collected data are sent. The three modules of the REMOA middleware were refactored as the following:

- Transparent Proxy module tasks are performed by AP and service servers. AP forwards packets based on flow rules sent by ThingsFlow application through the controller; data collected by *things* are sent through an IPSec tunnel to the services where are parsed and processed.

- *Things* management that was mostly based on SNMP in REMOA middleware, now is based on OpenFlow counters. Counters are retrieved from APs by ThingsFlow that makes it available with a timestamp indicating when it was retrieved; services retrieve counters stored in ThingsFlow to implement management mechanisms to its *things*.

- Gateway now is the packet forwarding feature realized by AP. In this approach forwarding is defined by OpenFlow rules. In current IoT scenario there is a trend to use IP-based approaches in *things*, such as 6LowPAN and CoAP, for example. If *things* communication is not based on IP, usually there is a device such as a gateway/hub that connects *things* in an IP network, so that packets are forwarded as any IP packet.

After the refactoring, the development needed for each service (communication with *things*, data collecting, processing, etc.) is not more impacted by AP constraints; developers can use any programming language, library, framework or programming techniques to develop features for services, specially those complex or even impossible to be developed to run on AP.

In this work we extended middleware capabilities enabling it to provide multiple services in a single AP. Yiakoumis et al. [18] also explore the sharing of network infrastructure by

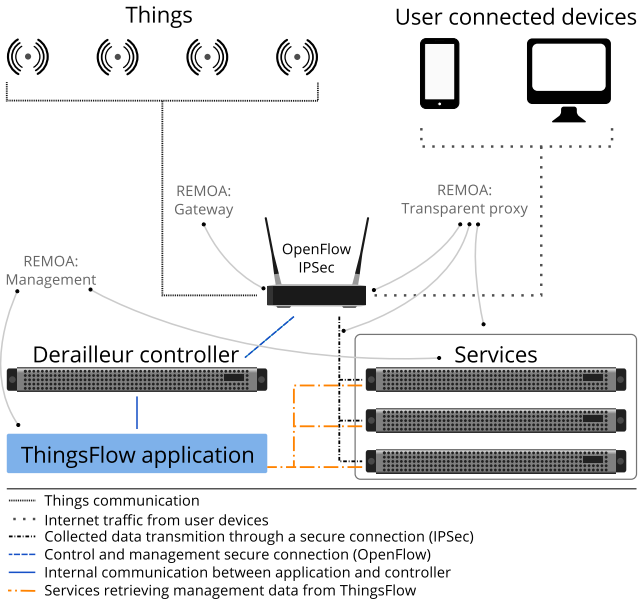

Figure 2: Middleware architecture refactoring.

different service providers, but in their work the network is sliced and each slice is isolated and is controlled by a service provider. Slicing does not apply to our scenario because we focus on IoT; in our approach services share data from their *things* enlarging the digital panorama of the physical environment. Due to privacy questions some data must not be shared, such as personal and health data of a patient; although data about *things* communication can be shared to be used by any service, even from those which collected health data.

Modules updates in APs in use in patients houses are no longer needed; AP, as an OpenFlow switch, receives forwarding rules on demand from the controller. IPSec settings are updated in each AP configuring them to create tunnels with servers running the services they are providing; it is done by a script that retrieves proper settings from the same server where controller and application are running.

The benefits provided by this approach are: the support to multiple services; the development of services and its features in a not constrained environment; and automated update of forwarding settings based on OpenFlow. In addition, SDN-based refatoring allows to recognize connected devices providing vision of network scenario and to implement management features suitable for IoT.

4.1 Derailleur Controller

The OpenFlow controller we developed in this work, called Derailleur, has been built on top of `libfluid` [2], the winner of Open Networking Foundation OpenFlow Driver Competition [8]. `libfluid` is divided in two libraries: `libfluid_base`, that implements a server to listen to switches connections and to handle switches events; and `libfluid_msg`, which creates, parses and sends OpenFlow messages. Derailleur controller was implemented extending `libfluid_base OFServer` class through inheritance.

Derailleur was designed to be lean and to abstract APs as switches objects, as shown in Figure 3. Derailleur retrieves information from each connected AP sending `OFPT_FEATURES_REQUEST` and `OFP_MULTIPART_REQUEST` (`OFPMP_DESC` to request description) messages; APs replies are parsed and stored in the corresponding switch object stored in a virtual stack (array), and managed by controller. Each AP may be recognized through the information retrieved; in our scenario we used MAC (Media Access Control) as an identifier; we can not use IP addresses because APs in home networks will receive IP address dynamically, additionally they may be behind NAT (Network Address Translation).

Derailleur runs an application (a child object of `Application` class) over their; Derailleur forwards events from APs to the application that will handle then according to the service, as described in Section 4.2. `Application` class provides four abstract methods: `on_switch_up`, `on_switch_down`, `on_packet_in`, and `message_handler`; controller triggers the suitable method depending on the AP event. When an event occurs, the controller sends to the application data related to the event triggered by switch; the application handles the event and communicate with the AP through a switch object stored in the switches stack. Switches stack is managed by `Controller` class and shared with the `Application` class. Through the switches objects, the application can contact APs to retrieve data (*e.g.*, flows, counters, information about connected devices) and to manage flow-tables.

To deploy OpenFlow in APs we used the virtual switch Open vSwitch (OVS) [3] on OpenWRT. We built an OpenWRT image with OVS, the resulting image also contains scripts to configure OVS bridges and network interfaces when AP boots up.

4.2 ThingsFlow Application

ThingsFlow extends Application class from Derailleur controller. ThingsFlow implements the inherited abstract methods `on_switch_up`, `on_switch_down`, `on_packet_in`, and `message_handler`; these methods are triggered by controller when AP events occur. ThingsFlow also provides methods to manage flow-tables in APs, independent of APs events.

`on_switch_up` runs always when an AP connect to the controller; ThingsFlow identifies the AP through the switch object stacked by the controller and queries database for flow-tables of services being provided on that home network and installs them on AP. The method `on_switch_down` is triggered when any AP disconnects from the controller; Derailleur removes the corresponding switch object from the stack; through this method ThingsFlow knows which AP disconnected and will not try to access it.

Flow-tables sent by ThingsFlow do not contain default rule to process packets from unknown devices (*e.g.*, devices not specified in any service). When a packet does not match any rule on the flow-table of a given AP, it causes a `table-miss` event making controller to trigger `on_packet_in` method; ThingsFlow installs flows to forward packets from that specific unknown device to the Internet or inside the network. Flows defined specifically for unknown devices enables the AP to implement flow counters for each connected device; the counters may be used by any service, as shown in Section 5, and provide a view of the network scenario. ThingsFlow also makes available to the services flows counters retrieved from APs; counters show if and when devices communicate and allow services to monitor their *things*.

Table 1 provides a basic example of flow-tables arrangement and packet processing in a scenario where a health monitoring service (REMOA) is provided. Beyond REMOA flow-tables, there is one table for packets pre-processing and

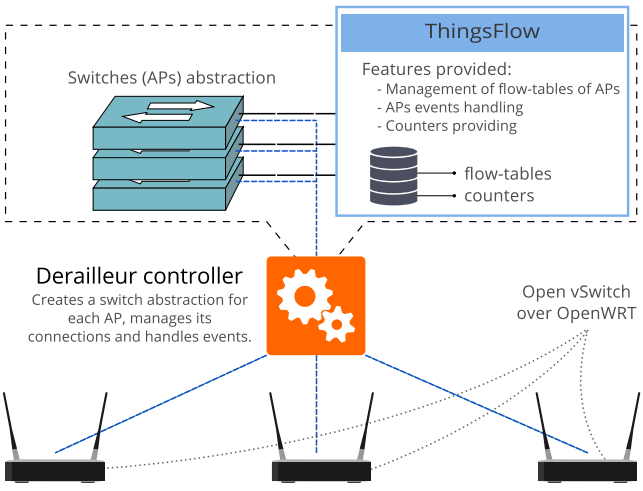

Figure 3: Overview of middleware components.

Main (0)	Input (1)	Output (2)	Firewall (3)	REMOA (4)
* Packets addressed to AP: go to table 1; * Packets sent by AP: go to table 2; * Packets from healthcare devices: go to table 4; * Packets to healthcare devices: go to table 4; * Packets addressed to healthcare manufacturer servers: go to table 4; * Any other packets: go to table 3.	* Check if the incoming packet is allowed or not in this device.	* Check if the outgoing packet is allowed to be sent by this AP.	* Check if packet is allowed according to the network and services policies; * Packets that are not explicitly refused causes a *table-miss* event; a *packet-in* message is sent to the controller.	* HTTP packets from healthcare devices are redirected to REMOA service; * Packets from REMOA service to healthcare devices are allowed; * HTTP(s) packets to healthcare devices' manufacturer sent by other devices are normally forwarded; * ICMP and SNMP packets to devices that support this protocols are allowed; * Any other packet is dropped.

Table 1: Example of packet processing among flow-tables.

three other providing filtering rules to protect AP and home network.

5. EXAMPLE SCENARIO

In this section we present some examples of services that may be provided in the same home network through the same AP. In the example scenario APs are supplied by local government to be used in social and/or well being projects provided, for example, by hospitals or universities. APs and ThingsFlow are managed by government staff that work together with services managers when services are deployed, configured or updated; for example, flows from different services may match the same packet, so managers need to handle this kind of issues, otherwise OpenFlow switches will not install overlapping flows.

This section describes three example services being delivered in the same home network. The following depicts how services could be deployed based on our approach:

Health monitoring (REMOA):

Provided by a hospital, this service monitors patients with chronic illnesses (*e.g.*, diabetes, hypertension) that do not need to be hospitalized. Patients are monitored through healthcare devices connected to the Internet through an AP using the existing Internet link in patient house. Healthcare devices and the AP are provided by the hospital.

Healthcare devices we used are hard-coded configured to send reading data to its manufacturer servers, so traffic from these devices are redirected to REMOA by OpenFlow that rewrites destination IP in packet header.

Data sent by healthcare devices provide a vision of patient health. ThingsFlow knows every device connected to the AP, it makes available flow counter of every device enabling healthcare application to draw a panorama of the entire networked environment and to infer about what is happening on patient's surroundings. This view of the environment as a whole is useful, for example, if a sensor does not detect patient presence for a large period of time, what could trigger an alarm, but their smartphone and notebook are connected and TV just started to transmit a streaming; probably patient is watching TV making soft movements that are not detected by sensor, although they are not unconscious.

Most of healthcare devices only connect when they are used for measurements and this must be done in pre-defined periods of time. Based on OpenFlow counters healthcare application may implement management mechanisms to suit treatment particularities, such as measurement schedule.

Access to smart city services

City government offers websites and mobile applications to be used by people; information available refers, for example, to transit, bus itinerary and current location, touristic places, events schedule. APs providing services in homes closer to the street, in special near bars, cafes or squares, broadcasts a SSID (Service Set IDentifier) to be used by anyone to access the city services; access attempts to any other content are dropped. The traffic of this service is entirely isolated from home network.

Research to characterize Internet usage

It is a research developed by an university to characterize Internet usage considering aspects such as people age, gender, education. OpenFlow counters allow to distinguish traffic from each connected device; it is made through specific flows that forward packets from devices of people participating of study to analyzed destinations, such as social networks, news, mail, web access, chat. Researches also consider time spent in each destination and period of the day, among others. Crossing people information with data from counters researches can define different usage profiles.

In the examples above we showed how different services could be implemented sharing the same network infrastructure, configured through a central application, the ThingsFlow. We emphasize that connected devices are not closed

in a single service; data from OpenFlow counters related to communication of networked devices allow that even *things* closed in manufacturer silos be used by different services for different purposes. Note that devices used by the third service example are the same that provide the panorama of surroundings of patients monitored; connections using the second service allows to estimate average waiting time in a bus stop or people concentration.

6. FINAL CONSIDERATIONS AND FUTURE WORK

In this paper we presented the refactoring process based on SDN of the REMOA middleware; we redesigned middleware architecture aiming to speed up development and deployment process of services, features and *things*. Our work focus on IoT services delivering in home networks.

Through the refactoring, network and service complexity that were concentrated on APs were distributed among different servers. APs were turned in OpenFlow switches in charge of packet forwarding according flows rules received from controller; they also provide a encrypted connection through which sensed data are sent.

The middleware were extended to provide multiple services through the same AP, in the same home network. As this work focus on IoT scenarios, *things* belonging to a service may provide useful data to other services; while users consume a service their devices connections may transparently supply data to the service itself, or any other service.

In our approach, all services run in remote servers provided with more computing resources than APs; there is no need to develop modules to run on APs. Developers can make use of any programming language, libraries, frameworks, and many other development resources that potentially make services deployment, improvements, and maintenance faster. Flow-tables of each AP are updated by controller when *things* and other connected devices communicate; no manual intervention is required.

Management of *things* must be designed considering *things* operating mode and their usage; OpenFlow counters allow to know when and how *things* are working. ThingsFlow periodically retrieves counters from APs and provide them to the services that implement management features to suit particularities of *things* they use.

Even running different services in the same AP we did not evaluated the impact of flow-tables processing in AP; it is one of the next steps in our work, as well a circumvent to deal with possible controller faults. Derailleur and ThingsFlow are both under development to finish some features; we intent to continue developing then to investigate other ways SDN can be explored in IoT scenarios and services delivering.

7. REFERENCES

[1] IOT-A: Internet of Things Architecture. `http://www.iot-a.eu/public`.

[2] libfluid - The ONF OpenFlow driver. `http://opennetworkingfoundation.github.io/libfluid/index.html`.

[3] Open vSwitch. `http://openvswitch.org/`.

[4] OpenWRT. `http://openwrt.org`.

[5] REMOA - Rede Cidadã de Monitoramento de Ambiente Baseado no Conceito de Internet das Coisas. `http://remoa.tche.br`.

[6] Software-Defined Networking: The New Norm for Networks. `https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf`, 2012.

[7] Silicon Labs 2013 Annual Report. `http://files.shareholder.com/downloads/ABEA-39NRLI/3514356491x0x730508/7FEFCB63-3F64-4D05-86BE-A9724E4AA891/SLAB\_32633\_proof\_rev2.pdf`, 2013.

[8] CPqD Selected as Winner and Recipient of $50.000 Grand Prize. `https://www.opennetworking.org/component/content/article/26-news-and-events/press-releases/1431-open-networking-foundation-announces-openflow-driver-contest-winner`, 2014.

[9] L. Atzori, A. Iera, and G. Morabito. The Internet of Things: A survey. *Computer Networks*, 54(15):2787–2805, 2010.

[10] M. Chetty and N. Feamster. Refactoring network infrastructure to improve manageability: a case study of home networking. *ACM SIGCOMM Computer Communication Review*, 42(3):54–61, 2012.

[11] L. Coetzee and J. Eksteen. The Internet of Things - promise for the future? An introduction. In *IST-Africa Conference Proceedings, 2011*, pages 1–9, 2011.

[12] P. Hinssen. The age of data-driven medicine. `http://datascienceseries.com/assets/blog/The_Age_of_Data-Driven_Medicine.pdf`, 2012.

[13] Linux Foundation. Technology Leaders Establish the AllSeen Alliance to Advance the 'Internet of Everything'. `https://allseenalliance.org/announcement/technology-leaders-establish-allseen-alliance-advance-internet-everything`, 2013.

[14] D. Miorandi, S. Sicari, F. D. Pellegrini, and I. Chlamtac. Internet of things: Vision, applications and research challenges. *Ad Hoc Networks*, 10(7):1497–1516, 2012.

[15] Z. Qin, G. Denker, C. Giannelli, P. Bellavista, and N. Venkatasubramanian. A Software Defined Networking architecture for the Internet-of-Things. In *Network Operations and Management Symposium (NOMS), 2014 IEEE*, pages 1–9, 2014.

[16] L. M. R. Tarouco, L. M. Bertholdo, L. Z. Granville, L. M. R. Arbiza, F. Carbone, M. Marotta, and J. J. C. de Santanna. Internet of Things in Healthcare : Interoperatibility and Security Issues. In *IEEE International Conference on Communications, International Workshop on Mobile Consumer Health Care Networks, Systems and Services*, pages 6121–6125, Ottawa, 2012.

[17] G. Wu, S. Talwar, K. Johnsson, N. Himayat, and K. D. Johnson. M2M: From mobile to embedded internet. *Communications Magazine, IEEE*, 49(4):36–43, 2011.

[18] Y. Yiakoumis, K.-K. Yap, S. Katti, G. Parulkar, and N. McKeown. Slicing Home Networks. In *Proceedings of the 2Nd ACM SIGCOMM Workshop on Home Networks*, HomeNets '11, pages 1–6, New York, NY, USA, 2011. ACM.