Conception of ID layer performance at the network level for Internet of Things

Jordi Mongay Batalla · Piotr Krawiec

Received: 24 November 2012 / Accepted: 21 February 2013 / Published online: 28 April 2013
© The Author(s) 2013. This article is published with open access at Springerlink.com

Abstract This paper proposes a novel architecture of the ID (IDentifier) layer for Internet of Things, which is embedded in the network level instead of traditional overlay solutions. Our contribution characterizes development of human-readable, hierarchical ID-based unified addressing for both objects and services, corresponding to their locations. In this way, users gain easier access to the IoT resources and native support for multicast. Furthermore, we take advantage of the Networking Named Content approach to specify rules for ID-based data transfer. The network nodes have capabilities to cache forwarded data for handling future requests, what may decrease network overload and facilitate cooperation between applications and sensors which periodically move into sleep mode for saving energy. ID-based routing offers decoupling of identification of objects/services from their location. Awareness of forwarded IoT data together with hierarchical distribution of the network makes feasible the local management of users and objects, that is, the essential IoT processes such as object/service registration, publication, searching and resolution can be managed locally in the network node. This offers high flexibility and manageability and improves response of the system when the number of handled things scales. The paper presents the architecture of the proposed solution for ID layer focusing on modules and mechanisms of the ID network node, as well as detailed description of registration, resolution and forwarding processes. Furthermore, the implementation of the system, performed on Linux-based routers, was tested

to confirm the correctness of ID layer processes. The tests show that the performance of ID network node is not worse than the performance of a classical IP Linux router running on an identical physical device.

Keywords ID-based routing · Internet of Things · Networking Named Content · Name data networking · Future Internet

1 Introduction

This paper presents novel architecture and mechanisms of Identifier (ID) layer for IoT purposes. Such a solution solves the registration of things (also called objects) and services as well as the search and delivery of the information related to them. We propose to move the ID layer to the network level, avoiding overlay solutions and offering network facilities as, among others, caching of data for energy saving, multicast of services and ID/location separation.

Moreover, we introduce a hierarchical addressing of objects following their physical location which contains the services offered by objects or groups of objects. The requests of data by the applications as well as the information from objects and/or services (data) are sent by the appropriate hierarchical tree. The nodes in the tree are able to cache the data and store for further requests during the validity period of the data. During this time, new interests will be served from the closest node that cached the data. This way, the request and data frames do not need to transfer the whole network (avoids overload) and data are provided in a short time. Moreover, sensors may go on sleep mode for saving energy consumption, whereas sensing data are still available for applications.

J. Mongay Batalla (✉) · P. Krawiec
National Institute of Telecommunications, Warsaw, Poland
e-mail: jordim@interfree.it
URL: <http://www.nit.eu/z3>

The most outstanding features of the proposed system are:

1. to integrate addressing of objects and services (thanks to human-readable hierarchical naming),
2. providing ID/locator separation with one unique addressing structure (thanks to the properties of concatenated IDs),
3. distributing and facilitating general process as registration and publication of objects/services (thanks to location-specific characteristics of IoT objects and services),
4. improving flexibility in multicast/anycast communications at different levels (thanks to provided ID-based routing),
5. improving forwarding of frames (thanks to the development of the mechanisms based on Networking Named Content—NNC approach),
6. energy saving of objects (thanks to the caching and storing of data in the network).

Let us remark that such an ID layer solution fulfills several main goals of the future Internet for Network Awareness specified in [1]. Specifically, it assures service diversity, functional flexibility, energy consumption and aspects of network management as self-organization.

The solution integrates different approaches, from IoT-specific as ID layer concept to network-specific ones as NNC mechanisms. Therefore, an extensive state-of-the-art is presented in the next section that stresses specially the most innovative visions in the framework of different research projects of these last years. After that, we present the general concept of the ID layer solution, and consecutive sub-sections show in detail ID routing and ID layer architecture. Section 4 presents implementation details of the prototype. The proposed solution with the mechanisms and protocols has been developed by using Linux routers and the final implementation has been validated and tested from the point of view of scalability (Sects. 4.1, 4.2). At last, the paper is summarized and further work is presented in Sect. 5.

2 Context and rationale

As referred in the CASAGRAS project, “Internet of Things (IoT) is a global network infrastructure, linking physical and virtual objects through the exploitation of data capture and communication capabilities” [2]. Briefly speaking, IoT is a huge connectivity platform for self-managed objects. A key challenging question in IoT research is how to identify and access the objects and to bind objects/resources to machine addressable and identifiable names, as outlined in most IoT-related projects, for example, [3–6]. These

functionalities are grouped in the so-called ID layer, which should be a convergence layer for all IoT technologies and it should be separated from the IP layer in order to avoid the limitations of IP addressing structure (e.g., no context or location awareness).

Several current projects focus on the definition of the IoT architecture, outlining the importance of the ID layer. IoT-A (Internet of Things—Architecture) [7] is a flagship FP7 project, whose main role is to develop an architectural reference model for the interoperability of Internet of Things systems. According to IoT-A, the main strength of the proposed communication model is the interoperability between heterogeneous networks. IoT-A provides a reference communication stack where, besides typical layers, there are two new ones [7]:

- ID layer: The virtual-entity identifier (VE-ID), split from the locator, is the center of the first convergence point in the communication stack, that is, the ID layer. Also, security, authentication and high-end services will exploit this layer for providing uniform addressing to the many different devices and technologies in IoT networks.
- End-to-end layer: This layer takes care of translation functionalities, proxies/gateways support and of tuning configuration parameters when the communication crosses different networking environments. By building on top of the ID and the network layers, the end-to-end layer provides the final building block for achieving a global machine-to-machine (M2M) communication model.

As far as end-to-end layer is application dependent, the ID layer is common to all IoT applications. One of the main challenges of the ID layer is the ID naming solution. There are two approaches to ID naming: hostname and hostID. The difference between hostname and hostID lies in the functions that they fulfill. Hostnames are used for human-comprehensive business (e.g., URL naming [8]), whereas hostID is for machine communication [9]. Translation between them is required (e.g., Domain Name System). Since hostIDs are the basis of routing, they have a fixed length, which is not required for hostnames [9]. In our solution, we propose to consider hostIDs which have the capabilities of hostnames, that is, they are human comprehensive.

The separation between IP and ID layers considers two possible approaches. The first one is studied in Veil-VIRO [10] and MobilityFirst [11, 12] and assumes the introduction of a new layer, which provides uniform convergence that integrates and unifies routing and forwarding performed by layer 2 and layer 3. The second approach (Ubiquitous Code end-to-end system, ucode [5]) assumes ID naming inside of the application layer as well as

distributed database solution for registration and resolution of IDs.

Proposed Veil-VIRO concept [10] describes a uniform convergence layer (below network layer) to support a dynamic and heterogeneous (layer 2) network that is capable of connecting large number of diverse sets of physical devices including both Ethernet and non-Ethernet. Efficient “Below IP” communication is provided by an additional routing substrate—VIRO [13] uses a structured virtual id (vid) space for object address resolution, routing and forwarding. In VIRO, both physical identifiers (e.g., Ethernet MAC addresses) and higher layer addresses/names (e.g., IPv4/IPv6 addresses or flat-id names) are mapped onto vid address space. The vid for end hosts comprises two parts: the L-bit host-node vid part identifies the host-node (VIRO switch), which an end host is directly connected to; the second l-bit host vid part identifies the end host and distinguishes it from other end hosts attached to the same host-node.

Routing in VIRO is inspired by Kademlia-like DHTs [14]. The key difference with DHT is that this last assumes end-to-end connectivity, and utilizes the underlying IP layer for routing and look-up operations, whereas VIRO must build end-to-end connectivity by itself, using the (local) physical connectivity among VIRO nodes. This way, VIRO creates networks, which connect many heterogeneous devices with rich physical topologies (as in case of IoT) and provides high availability of these networks and the services running on top of them.

MobilityFirst [11, 12] introduces a global unique identifier (GUID) which is assigned to the object independently of its network address and location. GUID is a string composed from object owner’s public key and hash key calculated for given object, and it is used by network nodes for performing hybrid GUID/network address routing. GUID layer has flat structure and it is placed above the network layer. In order to maintain mapping between GUID and related object’s network address, the authors proposed a logically centralized Global Name Resolution Service (GNRS). MobilityFirst routers may deliver data taking into account GUID only or using GNRS to find network address related to given GUID. Furthermore, for searching purposes MobilityFirst requires external name assignment service to assign and publish a GUID with its semantic description.

Ubiquitous Code (ucode) [5] uses a different approach for IoT object identification since it is applied on application layer. The ucode is a 128-bit fixed length unique identification number that unites virtual objects with things. There is no relationship between the numbers itself and the attribute and meaning of the target to which the ucode is assigned. A ubiquitous ID system basically stores information that represents attributes and meaning of the target

in databases (resolution servers with hierarchical structure). The system assumes that an intelligent terminal equipped with the ucode client library and connected to resolution servers’ infrastructure reads the ucode.

There are many other proposals for solving the problem of naming and addressing in the Internet of the future. It is generally accepted that the future networks should provide novel addressing for new pushing business models as IoT. Several solutions for ID-based addressing frames are being currently proposed in Content Centric Networking (CCN) [15], which is the prelude of Named Data Networking (NDN) [16] and Content Aware Network approaches as DONA [17] and CURLING [18].

Networking Named Content (NNC) approach, considered in CCN and NDN systems, implements routing based on content IDs, which are not necessarily hierarchical. Simply, the frames go through the route toward the possible content provider to “find” the requested ID. If the content is already found in intermediate caches, it is delivered back to the requester. This feature is interesting for IoT, particularly due to the behavior of small, battery powered sensors which may work at active mode and handle request from applications, or may enter into sleep mode turning off both the receiver and transmitter to save power. The proposed in-network caching allows storing the results of sensors in the network nodes, where the applications are able to read them in anytime, also during the period when the sensor is in sleep mode. This is especially convenient for scenarios assuming exploitation of many heterogeneous sensors, where each sensor may have individual energy saving scheme [19]. Therefore, we assume to use in-network caching functionality in our system.

Moreover, Networking Named Content supports plain text addresses that are directly human readable, and therefore, does not need intermediate services like DNS. The address of the content to be found can be limited to a local search as long as an ontology is defined (for example, /ThisRoom/printer/ is a simple “context-aware” routable content), which makes ease the discovery of the neighborhood and services. There are no further requirements about ID naming excepting the uniqueness of the domain name announced by each content provider.

DONA [17] uses flat, not human-readable and well-structured ID naming in order to route the frames within the network. The authors introduced a clear separation between resolution process and data delivery, which in practice requires overlay structure to resolve the requests. Similar solution assumed for CURLING [18].

The so-called ID/locator split concept [9] consists of considering machine location in IP addresses. Machine location has been a challenge since the beginnings of the Internet and has returned with the introduction of the IPv6. The challenge seemed feasible thanks to the large

addressing space, which could solve issues related to security [20] and multihoming [21]. The more advanced protocol for routing ID/locators is the LISP [22]. Actually, the proposition did not fit in because the current Internet is not prepared to distinguish close and distant communications. Nevertheless, this solution could work for IoT communication since this is very location dependent, that is, normally things speak with things that are close, whereas distant communications may be solved with overlay architectures. Because of this, we will consider some of the features of ID/locator split concept in our routing.

3 Solution for ID layer at the network level

The potential of the IoT lies in its ability to gather, analyze and distribute data [23]. The system proposed here fulfills the functionalities of the ID layer improving the gathering and distribution of data, since the data are stored within the network nodes near the applications (end users of these data).

The main idea consists of embedding the ID layer into the network level, offering awareness about IoT data to the network nodes. The transfer of IoT data is based on hierarchical network addressing, which allows a tree-based routing, that is, the data frames are transferred by established trees in both upload and download directions, as it is performed in NDN project [16]. Human-readable addressing of objects and services makes easier the programming and the access to multicast services.

On the basis of the hierarchical ID addressing, we propose an architecture for the ID layer that complies with the following phases of data distribution:

- Registration, publication and resolution of objects and services. The things (or objects) in the system should register in the closest network node and also register all the basic and composed services that they offer. Services may be a single action over one sensor (basic service) as, for example, switching on the light or it can be a group of actions on different sensors ordered by software (composed service). Then, the applications can search objects and services in the network or in one particular area of the network by asking to the appropriate network nodes, which will send information about the registered objects/services (resolution).
- Data distribution. Embedding of the ID layer at the network level makes possible that the network nodes are aware about the data transferring the network. We use this important feature in order to improve data gathering and delivery by introducing IoT-specific caching in network nodes. The data are closer to the

application side, saving response time as well as traffic load in the core network. Unlike the content data, IoT data expire after the validity time, which is an added parameter introduced for IoT purposes and depends on the type of data. Therefore, the objects can send the data to the closest network node periodically with an interval equal to the validity time, and then, the objects can go on sleep mode to reduce energy consumption. The validity time can be significantly higher than current sleep mode intervals: for example, in the case of temperature, the validity time of sensing data equals several minutes [24]. During this time, the network node will serve the applications' requests with the cached data.

The introduction of IoT awareness in the network is a new approach compared to other solutions [4–6, 10] that rather base on overlay architectures. In the same way, most of the solutions proposed in the literature for discovering and accessing the services in IoT are limited to overlay systems (e.g., [11, 12]).

Figure 1 presents an exemplary usage scenario of the proposed solution. The functionalities of the ID layer are in the network nodes, whereas objects and applications perform only encapsulation functionalities. Similar to ETSI machine-to-machine architecture [25], we distinguish two kinds of objects: (1) those which have implemented encapsulation mechanisms and can interact directly with network nodes and (2) those which connect with network nodes via IoT controllers. The second group covers different sensor area networks, composed with resource-constraint devices which communicate with the IoT controller using a given technology (Zigbee, Z-Wave, etc.). The

Fig. 1 Exemplary network scenario

controller is responsible for communication with the network nodes.

The connectivity with other networks is assured by the introduction of end-to-end layer, as defined in IoT-A (see Sect. 2). The end-to-end layer gateway is the responsible of providing end-to-end connectivity.

The approach of gaining IoT data awareness at the network level offers faster responses, security improvements, robustness and simplicity in accessing the services. Anyway, there are at least two issues that should be analyzed in our solution, which are: (1) the possible incompatibility with current networks and (2) scalability issues. The first issue is similar to the evolutionary approaches presented in Content Aware Networks [26, 27], where new ideas for improved content transfer were proposed. These ideas are consistent with current networks and do not need a revolutionary swing. The general approach is to connect the dedicated nodes by IP or MPLS tunnels obtaining a topology which is aware of the specific thematic. In the tunnels, the packets may be forwarded by other IP routers without performing added actions. This approach could be taken to implement the architecture proposed here in running IP networks. Scalability aspects will be analyzed further on in this text.

3.1 ID routing: definition, requirements and rules

We propose human-readable names of objects and services which are the basis of the network routing. Also, network nodes are labeled with human-readable IDentifiers. At last, the naming scheme will make use of hierarchical ID-based addressing scheme, which will be created and managed in conjunction with the location of each IoT object.

We introduce Multi-ID [28] naming for integrating IDs assigned to objects and services. An example would be an identifier, where we specify the order for the sensor of the bulb (switch on!). Specifically, the name of any node, object or service is formed by 8 ASCII-extended characters (a word). We use ISO 8,859-1 [29] as the extension of ASCII. The address of nodes, objects and services is the concatenation of all the names from the root “.”, separated by a dot. The character ASCII 2E₁₆, that is, the symbol “.”, should be reserved in naming in order to introduce hierarchy, which will be crucial in routing. The character ASCII 2A₁₆, that is, the symbol “*”, is also reserved for multicast/anycast calls. We limit the number of levels in the hierarchy (from the root to the furthest leaf) to 64, so the longest address in the network is 512 bytes. This way, we solve one of the scalability issues of the NNC approach, which is variable length and unbounded naming [16]. Moreover, hierarchical naming facilitates exact string matching, even for naming fast updates, since updates can be bounded and solved in a local area. Exact string

matching is also one of the main scalability issues in NDN project [16].

The maximum length of the payload field of an Ethernet frame is 1,518 bytes, which allows for 1,000 bytes for data, which is more than enough for the vast majority of IoT applications. An example of the ID is shown in Fig. 2.

The ID is written in a dedicated header of each frame together with information about the header length. This makes it possible to understand the actions that must be executed on the forwarding nodes only by comparing the ID with the address of the node.

ID-based routing offers new possibilities for IoT environment. It decouples identification from location, enabling the user to access IoT resources in flexible and convenient way. Particular names (words) of an ID indicate a virtual entity we want to communicate, whereas the ID as a whole (i.e., the list of words) indicates the current localization of the entity. ID routing simplifies delivery of data addressed to group of objects on given area. Exploiting hierarchical capabilities of proposed naming scheme, we can send data to, for example, part of lights in floor001 sub-domain using the ID of the form: floor001.*.lightctr.l1-sw_on, see Fig. 3a.

Frames are delivered to the edge switch at a given sub-domain, where they are multiplied and sent to the destination controllers or sensors. A frame with address “.*” is a broadcast frame which arrives to all objects and services. Anyway, being accessible by a multicast request depends on a given object or service. For example, switching off an emergency light should not be allowed for multicast calls but only for unicast. The protocol used for communication introduces this issue as we will see in next chapter.

The proposed ID-based routing offers higher efficiency thanks to the fact that end-to-end functionalities (location, identification) have been put into the network [30]. Moreover, such routing gains in flexibility since it offers multicast in all the physically located levels (e.g., send to all the lights in one room, or send to all the objects of one floor, or send to all the computers of the enterprise). In the case of IP routing, we should address packets separately for

Address fixed words	Level of hierarchy
floor001	1
room0001	2
lightctr	3
l1-sw_on	4

ID = floor001.room0001.lightctr.l1-sw_on

Fig. 2 ID naming example

Fig. 3 Differences between IP- and ID-based routing

each controller, therefore, distinct flows should be transferred between source and destinations, see Fig. 3b. Although IP multicast can be used to improve network utilization, such solution fails if we take into account the possible high dynamicity of groups of things we want to communicate with (IP addresses do not offer the dynamicity of ID names).

3.2 ID layer architecture

Clear communication rules are required at the ID layer in order to provide communication between IoT system components, that is, sensors and executors, software agents, human-adapted interfaces, etc. These rules should comprise registration (new objects and services register in the ID layer), publication (the registered objects/services are publicized for access), resolution (software agents discover the published objects/services) and data request (object or service is required) actions. Moreover, data delivery has their own rules, which are adopted from NNC but adapted for IoT purposes. All the actions over objects (e.g., setting parameters in sensors or executors) and services (actions to be performed by the objects) are handled in the same way by the ID layer.

The proposed scheme for ID layer architecture is presented in Figs. 4 and 5. Each node in the network contains

elements fulfilling functionalities of each one of the different planes of the architecture: *Resolution and reachability* plane discovers the objects/services for users (other objects, human adapted interfaces, software agents, etc.) and manages the reachability of the registered objects/services; *Naming and registration* is responsible for assigning unique hierarchy-based and well-structured IDs to the objects and services; and *Data forwarding* which is aimed at providing the forwarding/routing rules for IoT frames as well as serving arrived requests by using cached data. Thanks to the fact that all the network nodes have interfaces for each plane and thanks to the hierarchical distribution of the network, the actions (registration, publication and resolution) are managed only locally, which offers high flexibility and manageability and improves the response of the system when the number of handled things scales.

In the following paragraphs, we describe in detail the running of the network node avoiding the security and privacy functionalities that are independent of the presented solution for the architecture.

Once a new object is added to the network or a new service is created or composed, they should be registered at the ID layer. For this, the object (or controller) or the service creator sends a *Register* message to the node to which they are attached (see Fig. 5). This message, presented in Fig. 6, is sent to reserved address *.localhst* and

Fig. 4 Architecture of ID layer—functional blocks

Fig. 5 Architecture of ID layer—global view

contains the name and information about the object/service (the information is 320 bytes long, at the most—this limit results from *Resolution_response* message structure, see Fig. 7). The network node receiving such a message first checks the name of the object or service (ID naming module—see Fig. 4). If there is another object or service with the same name or if the name contains a reserved character (“.” and “*”) or reserved word (“localhst”), then it returns a *Register_failed* message with the parameter “name not permitted”. Otherwise, the frame is accepted and the name and information about the new object or service is stored in the Object/Service register (publication action).

In all the IoT messages, there is a field called Message info which is 2 bytes long. The first four bits define the message type, the next bit identifies whether the message is multicast or not (“1” = multicast; “0” = unicast). Finally, 11 bits indicate the length of the message in bytes.

The registration and publication are performed only in the first network node, so the message is not forwarded to other nodes. Whenever the bit multicast of the *Register* message is set to 0, then the Registration module should notify it to the Unicast only module, which stores the objects/services that cannot be accessed by a multicast request. The Unicast only module avoids future forwarding of multicast messages to this object/service. At the end of the Registration action, a *Register_accepted* message with the complete address of the new object/service is sent to the port by which the *Register* frame arrived.

The total address is constructed by concatenating the address of the network node and the name of the new object/service. The number of concatenated identifiers making up the address is written in the field Addr. length (max. 64). Longer addresses are not allowed and, in this case, registration should be rejected (*Register_failed* with the parameter “address too long”). The module Registration should

Fig. 6 Message structures for registration process

Fig. 7 Message structures for resolution process

notify the problems with addressing to the module Routing rules.

The actions performed by the Routing rules module should be taken in a central way and should consist in a re-ordering of nodes in the tree in order to obtain a tree with 64 levels to the push or re-adjustment of routing in the network. The introduction of new nodes should also be managed by the Routing rules module. This paper does not deal with these issues since they are not essential for the normal operation of the system.

The information about the new object/service, together with the port where it is attached to and the multicast flag, is stored in the IoT forwarding table at the *Data Forwarding* plane.

Once the objects and services are registered and published in the Object/Service register, the applications can discover them in the *Resolution and reachability* plane for composing new services. The application terminal sends a *Resolution* message (see Fig. 5) to required node which responds by sending a series of *Resolution_response* messages, one for each object and service registered in the node. The *Resolution_response* message contains (see Fig. 7) the address of the object or service and the information about the object/service. Moreover, the network node sends to the application other *Resolution_response* messages, each one containing the address of each child node. With these addresses, the application can find all the objects and services which are in the domain of the network node.

Note that another way to access to the child nodes' objects and services is by sending a multicast *Resolution* message, which can discover all the services of, for example, one floor or one room. In the same way, the *Resolution* message is a broadcast message when the destination address is the root address (the root address has Addr. length equal to 0 and no Address). From the point of view of the application, the resolution is performed locally since the applications can know where the requested object or service is located (e.g., in floor1.room2). In our system, we use this characteristic, which is inherited from IoT features. Anyway, some objects or services may have a wider localization (it can be in the building or in the whole enterprise) and, in this case, the process of discovery should be performed at this wider (however, always local) level. This feature should be considered for solving mobility issues.

The lowest plane of the architecture is the *Data forwarding* plane. For forwarding the frames, the first field checked in the frames is the Addr. length (see Fig. 8) and the address itself. If the address is *localhost* or if the address corresponds to the network node address, then the frame is sent to the Registration or Resolution module, depending on the message type. In the case that the frame is not directed to the network node, then the forwarding decisions are taken within the IoT forwarding table by considering the following rules. If the address of the network node matches with the upper prefix of the destination address of the message (considering the "*" symbol), then the message should be sent to the child node, whose name coincides with the next level (hierarchical) of the destination address of the message. If this coincidence does not exist, then the message should be discarded. If the next level of the message destination address contains the symbol "*", then the frame should be sent to more child nodes by considering the information about the only unicast services and objects stored in the Unicast only module.

In the case that the address of the network node does not match with the upper prefix of the message destination address, then the message is forwarded to the parent node.

The network node performs added operations related to caching of messages for *Request* and *Data* messages. Such operations are performed in parallel and independently of forwarding operations in order to improve the performance of the network node. For the *Request* message, the network node checks whether the requested information is in the Data Caching Table. In this case, the request is served with the information from the table. This occurs when the following three conditions are accomplished: (1) the destination address in the message must coincide with an entry in the Data Caching Table list, (2) the validity time written in the Data Caching Table (and previously provided by the object/service) should be longer than the current time in the network node (we suppose timing synchronization between network nodes and objects that provide validity time of data; note that synchronization in the range of hundreds of milliseconds is enough; in our case we used Network Time Protocol), and (3) the validity time of the *Request* message and the validity time from the Data Caching Table are not equal. This is used to request refreshed information from the object/service.

If the *Request* message is not served from the Data Caching Table, then the message is cached in the Request Caching Table. Specifically, this table stores the port from which the *Request* frame arrived and the destination address of the object/service. This way, when the answer to the *Request* (*Data* message) arrives to the network node, the source address of the *Data* message (see Fig. 8) is found in the Request Caching Table and the frame is sent to the port pointed in the table. If the source address is not in the Request Caching Table, then the *Data* frame is discarded. The entries in the Request Caching Table are deleted when the validity time of the entry expires. If the application does not want that the *Request* message is cached in the table, then it may set the validity time of the message to 0. Other sanity operations may be provided in the network node in order to avoid the permanence of requests during a long time. This way, the system is able to easily handle large-scale number of flows [16] with limited hardware.

Fig. 8 Message structures for data forwarding process

Some *Request* messages do not expect a response, for example, an action ordered to an executor. For this purpose, the request messages have a Data payload field. These *Request* messages should set the validity time to 0 in order not to be cached in the Request Caching Table.

The *Data* messages are cached in the Data Caching Table, which stores the source address of the object/service, the validity time of the information and the data. The information is updated when another *Data* message with the same source address arrives to the network node. Moreover, the network node could perform sanity operations in order to delete old (after validity time) entries in the table.

As we can see in Fig. 8, the *Data* message is a unique message that starts with 112 (the others start with 002). These two bits are used to easily notify the forwarding node that this is a *Data* message, since the *Data* message requires different forwarding rules which are not based on the Destination Address.

The last message that we specified is the *Resolution_alert* message (see Fig. 7). In the case that an object does not respond to the software agent when it sent a *Request* message, then the software agent requesting the object/service can send a *Resolution_alert* message to the network node to which the desired service should be attached, that is, the parent node of the object/service. The software waits for an acknowledgment response from the network node and if the ack does not arrive, it can send a new *Resolution_alert* to the parent node of the last network node, and so on. The *Resolution_alert* aims at notifying the network nodes that an object/service is not active. The network node will try to contact with the object and, as a last resort, will unregister the object or service.

4 Implementation and test results

The objective of our implementation was to deploy the necessary modules and procedures to implement registration and resolution of IoT objects/services in addition to IoT data forwarding using an ID-based intra-domain routing. The layer 3 functionalities of the network node were developed from scratch on the basis of Ethernet (layer 2) infrastructure. For proper running, we defined a new Ethertype label (86d1₁₆) for IoT frames. All the frames arriving to the network node with such an Ethertype are handled by the ID-based forwarding rules.

The proposed network node has been implemented in a server with processor Intel® Core™2 Duo Desktop Processor E8500 3.16 GHz and Linux Kernel version 2.6.17. The modules related with data plane functionalities were implemented in the kernel space in order to assure effective data forwarding. The modules related with registration and

resolution processes were implemented in the user space and the communication between user and kernel spaces based on *libnl* library. The modules in the user space actualize the forwarding table that is in the modules located in the kernel space. Whereas, the modules in the kernel space delivers (to the user space modules) the messages related with the *Naming and registration* and *Resolution* and *reachability* planes and directed to the own network node.

4.1 Conformance tests

The final version of the implementation was checked by conformance tests. The revolutionary approach of the proposal assumes that all the network nodes, which mediate in the IoT traffic exchange, should be ID-forwarding aware. Therefore, we provided only conformance tests and not interoperability tests [31]. Anyway, the proposed testbed has the characteristics of interoperability testbeds, that is, the System Under Test (the IoT network node) and the reference equipment (software agents, sensors and executors, etc.) together defined the boundaries of the tests. Moreover, as in the case of interoperability tests, our tests were performed on interfaces that provide normal user control and observation. Figure 9 presents the testbed for conformance tests. This testbed contains one computer with an application for the control of lights (attached to .floor001.room0002) and 2 light controllers (in two rooms). These controllers mediate the transfer traffic from/to IoT system elements (i.e., sensors, actuators that are not intelligent) to/from the network. In our testbed, each light controller registers three services for each one of the two lights attached to the controller. The services are *sw_on* and *sw_of* for switching on and switching off the lights as well as *state* to obtain information about the state of the light. For example, the light controller in the first room of the first floor registers the services .floor001.room0001.lightctr.l1-sw_on, .floor001.room0001.lightctr.l1-sw_of and .floor001.room0001.lightctr.l1-state related to the first light in the node with address .floor001.room0001.lightctr (see Fig. 9).

The conformance tests are based on testing registration, resolution and delivery of IoT data by the network nodes. Moreover, we performed tests for checking correct forwarding of multicast messages. The correctness of the registration, resolution and delivery processes was confirmed by controlling the values of the different tables related to these processes inside the network node as well as by checking the frames (format and headers) arriving and leaving different physical ports. We used the Tcpcdump [32] program to gather network dumps, from which statistical graphs we plotted with the help of Wireshark protocol analyzer [33].

Fig. 9 Testbed topology for conformance tests

In order to perform the conformance tests, we initialized all the network nodes and configured them with the addresses shown in Fig. 9. Afterward, we performed the tests following the steps presented in Table 1. Table 1 offers also the results of the tests. We may observe that the prototype passed all the conformance tests provided.

4.2 Performance and scalability

We have discussed already about the better scalability of our solution in comparison to the NNC approach. In this section, we deal with IoT-specific scalability. The scalability is the ability of the system to efficiently accommodate continued growth either in size (expressed by the number of nodes and links) or in capacity (expressed by the number of users).

The ID layer solution proposed here is designed for local IoT networks as, for example, enterprise, home, public building, whose scale has a much lower range than the global Internet. Therefore, the scalability issues due to growth in size are not crucial. Whereas, the growth in capacity is a crucial factor in IoT because of the large amount of objects connected to the network.

The operations performed into the ID layer as presented in this paper can be classified into two families, depending on the time scale in which they are performed. The operations related with the registration, publication and resolutions are performed in a mid-term time scale, while the operations related with the frames transfer across the network are performed in a short-term time scale and are the real challenge for the performance of the system.

Registration and publication are executed locally, which solves the problems related with scalability since increasing number of objects requires more hardware only in a local range (e.g., memory for object/service register).

The most sensitive issue is the forwarding of frames when the number of objects and services increases. In these tests, we want to address the scalability analysis by understanding the forwarding characteristics of the implemented network node when the number of handled objects and services increases.

4.2.1 Performance tests

The first tests try to understand the behavior of the implemented network node by comparing it with the characteristics of a software IP router. The testbed considers one server (with installed IoT network node or Linux IP router) connected to the tester by two 1 Gbps Ethernet links in ring topology as recommended by the benchmarking methodology for measuring network interconnect devices [34]. We first checked the maximum throughput and frame loss rate for the implemented IoT network node and afterward, we installed a Linux IP router in the same hardware and repeated the tests. In both cases, the traffic was generated by Spirent TestCenter (equipped with CM-1G-D4 card). For IoT network node tests, the generated traffic was a flow of Data frames with an identical 24 bytes source address and validity time equal to 0. The size of the complete frame varied for different tests (64, 128, 512, 1,024 and 1,518 bytes) as proposed in [34]. When a *Data* frame arrives to the System Under Test (IoT network node), the source address of the *Data* frame is searched in the Request Caching Table and the frame is forwarded to the port recorded in the table. This port is the one connected to Spirent TestCenter in a ring topology. The *Data* frame is not cached in the Data Caching Table since the validity time is 0. For IP router tests, the frame lengths were the same as IoT node tests, but the frames contained normal IP packets. In this case, the forwarding rules were based on the IP routing table of the Linux IP router.

Table 1 Conformance test results of the IoT network node

Test category	Description	Steps	Expected	Actual	Result
Register	To register a new service in the closest network node	To send a <i>Register</i> message to port p1 of .floor001.room0001.lightctr	The <i>Register</i> frame arrives to port p2 of .floor001.room0001 node	The <i>Register</i> frame arrived to port p2 of .floor001.room0001 node	Pass
		To control p1 in .floor001.room0001.lightctr as well as p1, p2 and Object_service_register table in .floor001.room0001	The new service is stored in Object_service_register of .floor001.room0001 node	The new service was stored in Object_service_register of .floor001.room0001 node	
			The <i>Register</i> frame is not forwarded to p2 in .floor001.room0001	The <i>Register</i> frame was not forwarded to p2 in .floor001.room0001	
			The message <i>Register_accepted</i> with correct parameters arrives to port p1 of .floor001.room0001.lightctr	The message <i>Register_accepted</i> with correct parameters arrived to port p1 of .floor001.room0001.lightctr	
Register_failed	To try to register a new service with a mistaken name in the closest network node	To send a <i>Register</i> message (with the symbol “.” in the service name) to port p1 of .floor001.room0001.lightctr	The <i>Register</i> frame arrives to port p2 of .floor001.room0001 node	The <i>Register</i> frame arrived to port p2 of .floor001.room0001 node	Pass
		To control p1 in .floor001.room0001.lightctr as well as p1, p2 and Object_service_register table in .floor001.room0001	The new service is not stored in Object_service_register of .floor001.room0001 node	The new service was not stored in Object_service_register of .floor001.room0001 node	
			The <i>Register</i> frame is not forwarded to p2 in .floor001.room0001	The <i>Register</i> frame was not forwarded to p2 in .floor001.room0001	
			The message <i>Register_failed</i> with correct parameters arrives to port p1 of .floor001.room0001.lightctr	The message <i>Register_failed</i> with correct parameters arrived to port p1 of .floor001.room0001.lightctr	
Resolution	To obtain information about the objects and services registered in a remote node	To send a <i>Resolution</i> message (with destination address.floor001.room0001) to port p1 of .floor001.room0002.appl001	The <i>Resolution</i> frame arrives to port p3 of .floor001 node and is sent by p2 and not by p1 of this node	The <i>Resolution</i> frame arrived to port p3 of .floor001 node and was sent by p2 and not by p1 of this node	Pass
		To control p1, p2 and p3 in .floor001 as well as p1, p2 and Object_service_register table in .floor001.room0001	The <i>Resolution</i> frame arrives to port p1 of .floor001.room0001 node and is not sent to p2 of this node	The <i>Resolution</i> frame arrived to port p1 of .floor001.room0001 node and was not sent to p2 of this node	
			The node floor1.room001 sends to p1 one <i>Resolution_response</i> frame for each service (6 services) stored in Object_service_register	The node floor1.room001 sent to p1 one <i>Resolution_response</i> frame for each service (6 services) stored in Object_service_register	
			All the frames arrive to p1 of .floor001.room0002.appl001	All the frames arrived to p1 of .floor001.room0002.appl001	

Table 1 continued

Test category	Description	Steps	Expected	Actual	Result
Data delivery	To obtain data from the requested service registered in a remote network node	<p>To send a <i>Request</i> message (with destination address: floor001.room0001.l1-state) to port p1 of .floor001.room0002.appl001</p> <p>To control the ports in all the nodes of the first floor as well as p1 in .floor001.room0001.lightctr</p> <p>To control the Data Caching Table in all the nodes of the first floor</p>	<p>The <i>Request</i> frame goes through the following (and only these) ports: p1 of .floor001.room0002.appl001, p3 and p1 of .floor001.room0002, p3 and p2 of .floor001, p1 and p2 of .floor001.room0001 and p1 of .floor001.room0001.lightctr</p> <p>.floor001.room0001.lightctr sends a <i>Data</i> message with the requested information to the .floor001.room0002.appl001, this frame goes through the network by the same path (opposite direction) as the <i>Request</i> and arrives to the destination node</p> <p>The Data Caching Table of all the network nodes of the first floor store the information contained in the <i>Data</i> message</p>	<p>The <i>Request</i> frame went through the following (and only these) ports: p1 of .floor001.room0002.appl001, p3 and p1 of .floor001.room0002, p3 and p2 of .floor001, p1 and p2 of .floor001.room0001 and p1 of .floor001.room0001.lightctr</p> <p>.floor001.room0001.lightctr sent a <i>Data</i> message with the requested information to the .floor001.room0002.appl001, this frame went through the network by the same path (opposite direction) as the <i>Request</i> and arrived to the destination node</p> <p>The Data Caching Table of all the network nodes of the first floor stored the information contained in the <i>Data</i> message</p>	Pass
Data caching	To obtain data from the requested service which is stored in the Data Caching Table of the network node	<p>To send the same <i>Request</i> message as in the previous test to port p1 of .floor001.room0002.appl001</p> <p>To control the ports p1, p2 and p3 of .floor001.room0002 and the port p1 of .floor001.room0002.appl001</p>	<p>The <i>Request</i> frame arrives to .floor001.room0001 and is not forwarded ahead</p> <p>.floor001.room0001 answers with a <i>Data</i> message (with the requested data) that arrives to p1 of .floor001.room0002.appl001</p>	<p>The <i>Request</i> frame arrived to .floor001.room0001 and was not forwarded ahead</p> <p>.floor001.room0001 answered with a <i>Data</i> message (with the requested data) that arrived to p1 of .floor001.room0002.appl001</p>	Pass
Multicast	Once each light controller registered the services in the attached network nodes, it asks for all the <i>state</i> services in the network	<p>To send a <i>Request</i> message (with destination address: floor001.*.*-state) to port p1 of .floor001.room0002.appl001</p> <p>To control the ports in the nodes and controllers</p>	<p>The <i>Request</i> message is multiplied in each node and the multiplied frames arrive to all the controllers and all of them send two <i>Data</i> messages, for l1-state and l2-state services</p> <p>The node: floor001 does not forward the <i>Request</i> frame nor the <i>Data</i> frames to the port p1</p> <p>The <i>Data</i> messages arrive to .floor001.room0002.appl001</p>	<p>The <i>Request</i> message was multiplied in each node and the multiplied frames arrived to all the controllers and all of them sent two <i>Data</i> messages, for l1-state and l2-state services</p> <p>The node: floor001 did not forward the <i>Request</i> frame nor the <i>Data</i> frames to the port p1</p> <p>The <i>Data</i> messages arrived to .floor001.room0002.appl001</p>	Pass

Fig. 10 Performance test results for IoT and IP nodes: **a** throughput, **b** frame loss rate

The tests were repeated five times in order to calculate the confidence intervals. All the results have confidence intervals fewer than 20 % of the mean values at the 95 % confidence level. For clarity purposes, we did not present the confidence intervals in the figures.

In Fig. 10a, the throughput of both network nodes (IoT and IP) is compared with the theoretical load for the media (1 Gbps). The throughput is the maximum offered load forwarded by the device without any losses. More offered traffic than throughput causes more losses, which is presented in Fig. 10b. Sometimes the loss ratio is very low and it is not possible to appreciate the values in the figures. For example, the throughput of the IP node for the 64 bytes frame size is 5.3×10^5 frame/s, that is, 270 Mbps, see Fig. 10a. But the losses of the IP node for the 64 bytes stream are only appreciable in Fig. 10b at the range higher or equal to 500 Mbps.

As we can observe in the figures, the forwarding characteristics of IoT network node are better than common software for an IP router. This is because the IoT network node performs a very simplified set of operations on each

frame. Instead, IP router performs many operations for different functionalities (e.g., checksum operations).

We realize that the reliability of the results depends on the selected hardware hosting the IoT and IP nodes. The results would be different for any other hardware. Anyway, the merit of our studies lies in the comparison of forwarding characteristics of both the network nodes: IP and IoT. The comparison-based performance tests credibly show that the IoT network node has similar behavior as common IP router.

4.2.2 Tests for addressing scalability analysis of frame forwarding

The scope of these tests is to understand the influence of the increasing number of objects and services in frame forwarding. Specifically, a higher number of objects and services increases the size of the tables in the network nodes and this can reduce the forwarding performance of *Data* and *Request* frames in the network nodes.

We prepared a Request Caching Table with an increasing number of requested services, that is, from 1,000 to 100,000 requested services, in order to analyze the forwarding performance decrease. We generated a stream of *Data* messages (and afterward *Request* messages) in the Spirent TestCenter and calculated the throughput (as performed in Sect. 4.1) for different Request Caching Table sizes. The *Data* and *Request* frames were 64 bytes long.

When a *Data* frame arrives to the IoT network node, the source address of the *Data* frame is searched in the Request Caching Table and the frame is forwarded to the port recorded in the table. This port is the one connected to Spirent TestCenter in a ring topology. We set the *Data* validity time to 0, to avoid influence of caching the *Data* frame in the Data Caching Table.

When the *Request* message arrived to the IoT network node, the requested service was not found in the Data

Fig. 11 Throughput of Data and Request frames for increasing size of Request Caching Table

Caching Table. Thus, the message (after comparing its address with address of the network node) was forwarded to the output port, to which Spirent TestCenter was attached in a ring topology. Afterward, the requested service (the same one in all the messages) is rewritten into the Request Caching Table. This means that the entry was the same one for all the *Request* frames.

Figure 11 presents the throughput of *Data* and *Request* frames for different Request Caching Table sizes. As we may see, the throughput of *Data* frames lightly decreases for increasing number of entries in the table. The operations of the *Request* frames on the Request Caching Table are performed in parallel with the forwarding operations of the frames. Because of this, for small Request Caching Tables (until 5,000 entries), the throughput of *Request* frames is invariable of the Request Caching Table size. For larger tables, the writing operations on the table are the bottleneck of the forwarding process and, because of this, the throughput for larger Request Caching Tables decisively decreases with the number of entries. In our implementation, searching in the Request Caching Table is not optimized and uses a simple ASCII hash function.

As we can observe, the decrease in performance is logarithmical (notice that Fig. 11 is lin-log scaled), which is fewer than linear decrease; so, we can assume that no scalability issues are expected in forwarding for the range of table sizes tested. Let us note that current versions of OpenFlow tables can ensure up to 100,000 parallel flows [35]. The first tests on scalability showed good performance of the network nodes but only the future use of the proposed system will show whether the system can work in large-scale IoT scenarios.

5 Conclusions

The vision of IoT assumes that objects (or things) and services offered by them can be found and used by other entities in an automated way, without human interaction. Therefore, object and service identification is considered as one of the main challenges in the field of IoT, which can be solved by creation of a device-independent abstraction layer, the so-called ID layer, which aims to expose objects and their services.

The usually proposed approaches for the ID layer do not affect network technology and deploy the ID layer on top of it (Over The Top solutions). In this paper, we propose to create such a layer inside the network in order to achieve high efficiency and simplification for registration of objects and services as well as searching and delivering of the information related to them. This means that the nodes in

the network are aware of the generated IoT traffic that is transferring by the network. The capabilities of the network nodes allow to introduce some IoT-specific mechanisms to improve end-to-end communication between applications, sensors and services, without the interaction of persons. This way, machine-to-machine communication results in a greater ease and, at the same time, effectiveness of such transmissions increases.

Our solution assumes exploiting some characteristics of the Networking Named Content addressing scheme for both objects and their services, jointly with ID hierarchical routing. By using a list of concatenated IDs for forwarding frames, we attain an ID/locator separation with one unique addressing structure. Proposed ID-based routing improves flexibility of multicast communication, whereas in-network caching adapted from NNC makes feasible cooperation between applications and sensors exploiting energy consumption saving mechanisms.

To build a prototype, we developed the considered network node over the Linux operating system, which allows for separation of control and data plane in the network devices. We implemented data plane functionalities in the kernel space, whereas control plane functionalities were developed in the user space. The proposed implementation is based on standard IEEE Ethernet protocol, this way our solution can coexist with nodes implementing legacy protocol stack which are unaware of IoT traffic (e.g., IP). The condition for this coexistence is tunneling the IoT traffic between two consecutive IoT-aware network nodes using, for example, IP tunneling or MPLS techniques.

We provided tests of the prototype directed to demonstrate conformance of the implemented solution and to calculate performance of the network node for forwarding IoT frames also in the case of increasing number of objects/services (IoT-specific scalability). The performance tests demonstrated that the presented solution reaches similar (slightly better) results than current versions of software IP routers. On the other hand, increasing the number of IoT data flows reduces the forwarding performance of the network node but this reduction does not seem to be a problem for scalability because of its logarithmical trend.

Further works will cover, besides defining the routing rules and implementing the Routing rules module, development of protocol for addressing mobility in ID layer. This protocol will be based on communication between network nodes in order to manage re-addressing of messages to new locations during the streaming.

Acknowledgments The authors would like to thank Jérôme François and Thibault Cholez from University of Luxembourg for the fruitful discussions.

Open Access This article is distributed under the terms of the Creative Commons Attribution License which permits any use, distribution, and reproduction in any medium, provided the original author(s) and the source are credited.

References

1. Future networks: objectives and design goals, ITU-T Recommendation Y.3001, 2011
2. CASAGRAS Project, Final report: RFID and the inclusive model for the internet of things, 2009
3. SENSEI Project, Reference architecture, Deliverable 3.2, 2008
4. SWIFT Project, D207—final SWIFT architecture, 2010
5. Koshizuka N, Sakamura K (2010) Ubiquitous ID: standards for ubiquitous computing and the internet of things. *IEEE Pervasive Comput* 9:4
6. IoT@Work Project (Internet of Things At Work), WP 2—communication networks D2.1—IOT addressing schemes applied to manufacturing, 2010
7. Internet of Things Architecture “Project deliverable D1.2—initial architectural reference model for IoT”, 2011. Available from <http://www.iot-a.eu>
8. Berners-Lee T, Masinter L, McCahill M (1994) Uniform resource locators (URL). IETF RFC 1738
9. Kafle V, Otsuki H, Inoue M (2012) An ID/locator split architecture for Future Networks. *IEEE Commun Mag* 48:2
10. Sourabh J, Yingying C, Zhi-Li Z (2009) VEIL: A “Plug-&Play” virtual (ethernet) Id layer for below IP networking. 1st IEEE workshop on below IP Networking 2009 (In conjunction with IEEE Globecom 2009)
11. Jun L, Yanyong Z, Nagaraja K, Raychaudhuri D (2012) Supporting efficient machine-to-machine communications in the future mobile internet. In: Proceeding of the IEEE wireless communications and networking conference workshops WCNCW 2012, April 2012, Paris, France, pp 181–185
12. Jun L, Shvartzshnaider Y, Francisco J, Martin RP, Raychaudhuri D (2012) Enabling internet-of-things services in the mobilityfirst future internet architecture. In: Proceedings of the IEEE international symposium on a world of wireless, mobile and multimedia networks WoWMoM 2012, June 2012, San Francisco, USA
13. Guor-Huor L, Sourabh J, Shanzhen C, Zhi-Li Z (2008) Virtual Id routing: a scalable routing framework with support for mobility and routing efficiency. 3rd international workshop on mobility in the evolving internet architecture, MobiArch’08 co-located with SIGCOMM 2008
14. Maymounkov P, Mazières D (2002) Kademlia: a peer-to-peer information system based on the x or metric. In: Proceedings of IPTPS02, 2002
15. Jacobson V, Smetters D, Thornton J, Plass M, Briggs N, Braynard R (2009) Networking named content. In: Proceedings of ACM CoNEXT 2009. Rome, Italy, 2009
16. Yuan H, Song T, Crowley P (2012) Scalable NDN forwarding: concepts, issues and principles. In: Proceedings of the 21th international conference on computer communications and networks (ICCCN), Munich, Germany, August 2012
17. Koponen T et al (2007) A data-oriented (and beyond) network architecture. In: Proceedings of ACM SIGCOMM, Kyoto, Japan, August 2007
18. Chai WK et al (2011) CURLING: content-ubiquitous resolution and delivery infrastructure for next-generation services. *IEEE Commun Mag* 49(3):112–120
19. Azad AKM, Kamruzzaman J (2008) A framework for collaborative multi class heterogeneous wireless sensor networks. In: Proceedings of the IEEE international conference on communications ICC’08, May 2008, Beijing, China, pp 4396–4401
20. Moskowitz R, Nikander P (2006) Host identity protocol (HIP) architecture, IETF RFC 4423
21. Nordmark E, Bagnulo M (2009) Shim6: level 3 multihoming shim protocol for IPv6, IETF RFC 5533
22. Farinacci D et al (2009) Locator/ID separation protocol (LISP), internet draft, Sept. 2009; draft-ietf-lisp-05.txt
23. Evans D (2011) The internet of things. How the next evolution of the internet is changing everything, White paper. CISCO IBSG
24. ISTA (2013) Temperature report, available on-line at. Access 01 2013. <http://www.ista.org/pages/resources/TempRH.php>
25. Machine-to-Machine communications (M2M); functional architecture. ETSI Technical Specification TS 102 690 V1.1.1 (2011–10)
26. Begen A, Mongay Batalla J, Koong Chai W, Śliwiński J (2013) Multi-criteria decision algorithms for efficient content delivery in information-centric networks. *Annals of telecommunications—Annales des télécommunications* 68(3):153–165. doi:10.1007/s12243-012-0321-z
27. Mongay Batalla J, Chen Y, Begen A (2012) Optimization of decision process in network and server-aware algorithms. ISBN: 978-1-4673-1391-9/12 ©2012 IEEE. *IEEE Networks* 2012, Rome, Italy
28. Kafle V, Inoue M (2012) Introducing multi-id and multi-locator into network architecture. *IEEE Commun Mag* 50:3
29. International Organization for Standardization (1998) Information technology—8-bit single-byte coded graphic character sets—part 1: Latin alphabet No. 1”, ISO/IEC 8859-1:1998
30. Tschöfenig H, Arkko J (2012) Report from the smart object workshop. Request for comments 6574
31. ETSI TR 101 667 Methods for testing and specification (MTS); network integration testing (NIT); interconnection; reasons and goals for a global service testing approach. Technical report, ETSI, July 1999
32. tcpdump command-line packet analyzer. Official web site: <http://www.tcpdump.org/>
33. Wireshark network analyzer. Official web site: <http://www.wireshark.org/>
34. Bradner S, McQuaid J (1999) Benchmarking methodology for network interconnect devices. Request for comments 2544
35. McKeown N et al (2008) OpenFlow: enabling innovation in campus networks, White paper on OpenFlow system