Manage resource-constrained IoT devices through dynamically generated and deployed YANG models

Thomas Scheffler
Beuth-Hochschule für Technik
Luxemburger Str. 10
13353 Berlin, Germany
scheffler@beuth-hochschule.de

Olaf Bonneß
Deutsche Telekom AG
Winterfeldtstr. 21
10781 Berlin, Germany
olaf.bonness@telekom.de

ABSTRACT

This paper presents an experimental design/approach that allows the standardized management protocol NETCONF to handle dynamically changing networks found in the IoT and Home-Networking domain.

Management of such networks is challenging, because they usually grow out of spontaneous device assemblies, rather than an engineering blueprint. Network membership may be highly dynamic and the devices might only possess very limited computation and communication budgets.

It is our goal to develop methods and strategies for automatic device discovery and configuration maintenance in such networks. In our experiment we dynamically generate YANG data models and NETCONF RPCs from device profiles written in JSON and map these to configuration commands in the lightweight MQTT protocol.

CCS Concepts

•**Networks** → **Network management;**

Keywords

Constrained Devices; IoT; NETCONF; YANG, MQTT, Data Modeling

1. INTRODUCTION

The NETCONF network configuration management protocol [4] is posed as the successor of the Simple Network Management Protocol (SNMP). It currently has strong support from network equipment manufacturers, service providers and the IETF standardizations community. However, outside this target group fewer people know about NETCONF and how it could be utilized for their configuration management needs.

One target area that could benefit greatly from standardized network and device management is the emerging domain of resource-constrained devices called the Internet of Things (IoT). Device management in this domain is usually done via dedicated Web-interfaces or corresponding smartphone apps. Larger installations might use cloud-based solutions specific to a certain vendor or industry consortium[1,2].

Most IoT installations consist of devices with very limited resources. There have been considerations to specify a reduced NETCONF feature-set, called NETCONF Light [8], to align the protocol requirements with the available resources in the application domain. This particular effort has been abandoned and work currently focuses on using the Constrained Application Protocol (CoAP)[11].

This paper investigates a different approach how NETCONF management may be made compatible with current IoT installations. Instead of running NETCONF on the managed device itself, we implemented a NETCONF bridge, that represents the IoT device domain in terms of device management. In order to achieve this, the NETCONF bridge is able to translate between IoT specific protocols and NETCONF. Our prototypical implementation uses the MQTT protocol [7] as an example.

The NETCONF protocol uses the YANG language [2] to model configuration and state data. The current usage model, defined by the RFC, assumes fairly static, preestablished data models. The corresponding YANG schemas may be discovered [9], but are usually already known by the managed and the managing device prior to deployment.

In our proposed NETCONF bridge, no such prior knowledge exists. The managing devices are unaware about functionality that is added to their management domain by attaching new and previously unknown devices. It must therefore be possible to dynamically generate a YANG data model for the managed IoT domain from data inherent to this domain. This data model may be adjusted as the network and device composition changes over time. New devices may be added, other devices leave the network or become unreachable.

Our paper presents a prototypical implementation for such a bridging solution in order to answer the question what components are necessary for such a dynamic use of the NETCONF/YANG framework and to identify potential missing elements.

2. RELATED WORK

NETCONF assumes that a managed device plays the role of a server and will be managed by a connecting client speaking the protocol.

ANRW '17, July 15, 2017, Prague, Czech Republic

ISBN 978-1-4503-5108-9/17/07…$15.00

DOI: http://dx.doi.org/10.1145/3106328.3106331

[1] https://www.bosch-iot-suite.com/remote-manager/
[2] https://developer.apple.com/homekit/

This architectural style follows the Internet End-to-End paradigm, but means that a managed device needs to be reachable from the client and has the necessary resources to implement and execute the required protocol functionality. Sehgal et al. [10] have shown that this assumption might challenge certain intended IoT hardware platforms.

Effort is underway to lower the protocol requirements by utilizing the Constrained Application Protocol (CoAP)[11] to carry NETCONF protocol messages. There is ongoing work in the IETF to define a compact encoding for protocol data that is better suited than XML or JSON for use on constrained devices [12], [13].

However, nothing has changed in the underlying architectural model. NETCONF over CoAP still closely follows the client/server communication pattern underlying the REST architecture [5] of web services, where a client initiates the connection and requests functionality from a server. This is a proven communication pattern, that can scale very well when the server has the necessary resources to satisfy all client requests.

When the communication partner providing the service has only limited resources, a communication style following the publish/subscribe pattern might be a better fit. This communication pattern inserts a third entity into the communication path, which is acting as a proxy and decoupling communication endpoints. Messages are only exchanged between the endpoints and this central element, called a broker, thus shielding the internal (IoT) devices from direct Internet access. The broker may provide additional functionality, such as caching. Expensive authentication and encryption functions can be offloaded to the broker and the attack surface, in case of misconfiguration or denial-of-service attacks, is minimized.

One area, where publish/subscribe architectures shine is information push. Pushing data, such as state changes or notification events, to a number of subscribers requires application state at the server-side. Otherwise client applications have to resort to polling, which can be very inefficient if data changes infrequently and communication resources are limited.

CoAP tries to solve the problem of repeated polling through the introduction of a so called *observer pattern* [6]. A client registers itself with the server to be notified about possible changes at the server side. While this solves the problem of repeated polling, the server now has to maintain a list of observers. This burdens the constrained device with the problem of managing this list and sequentially notifying all observers about state changes, leading to potentially undesirable traffic patterns at the server-side.

Our experimental design uses MQTT for realising the publish/subscribe architecture, because it is an open protocol that has interesting properties suitable to our use case, good tool support and is widely used as a messaging protocol. It runs well on constrained devices and a MQTT client can send and receive messages for multiple topics over a single TCP-connection, therefore requiring very limited connection state.

3. MESSAGE QUEUE TELEMETRY TRANSPORT (MQTT)

Message Queuing Telemetry Transport (MQTT) is a lightweight messaging protocol for the connection of embedded devices that follows the *publish/subscribe* communication pattern for distributed applications. The protocol was originally developed by IBM and is currently standardized by Oasis [7]. It has been specifically designed to work in unreliable network scenarios, where data may be carried over low-bandwidth, high-latency links. MQTT requires very little resources from the participating devices. Implementations exist for all important computing platforms, including 8-bit microcontroller devices.

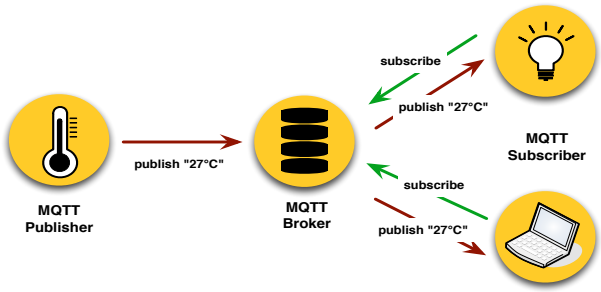

Figure 1: MQTT architecture consisting of Publisher, Message Broker and Subscribers

The communicating entities are called *Publisher*, *Message Broker* and *Subscriber* (cf. Figure 1). Publisher are the source of data. Message brokers queue, aggregate and distribute messages to subscribers. The message broker decouples the publisher from the subscriber. This an especially useful function in scenarios where the communicating entities have limited resources and become disconnected temporarily from the network, such as sensor nodes in an IoT use case.

Messages contain a so called *topic* in addition to the actual data. Topics are UTF-8 encoded strings that can be used to build a naming hierarchy similar to the path hierarchy in a file system. The hierarchy is build with the *topic level separator* '/'. The following example shows a topic that identifies messages from/to the telephone in Room 2 of House A:

```
House_A/Room_2/Phone
```

Each interested client that wants to receive messages for a topic subscribes to this topic from the message broker. It will then be notified by the broker when new messages arrive. Subscriber do not need to poll for new data, making it easy to scale the subscriber network without requiring more resources or complex configuration at the publisher. A subscriber can use wildcards to easily denote interest in messages for multiple topics. MQTT supports single-level and multi-level wildcards.

MQTT uses TCP for reliable, acknowledged message transport and can optionally use TLS to provide confidentiality and strong authentication. The protocol itself is data-agnostic. This means that the protocol assumes no structure for the transmitted data and a data publisher can transmit any form of digital data (ASCII, binary, etc.) that is understood by the subscribers. The broker usually forwards the message transparently and only processes the topic.

TCP guaranties reliable message transfer as long as a connection exists. However, in use cases with constrained

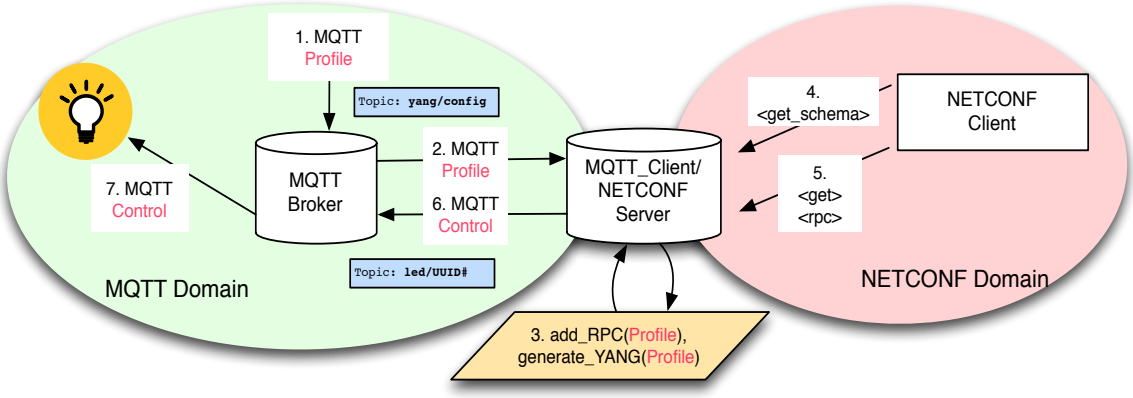

Figure 2: System architecture of the NETCONF/MQTT bridge.

devices, network connectivity may be unreliable. MQTT therefore provides a message acknowledgement mechanism at application level, called QoS that works in conjunction with a session abstraction that can span multiple TCP connections.

4. IMPLEMENTATION

Our prototypical NETCONF/MQTT bridge implementation makes dynamic IoT networks manageable via the NETCONF protocol. Such a solution requires three distinct functional components, a discovery mechanism, that allows the bridge to learn about available devices and their management capabilities. Secondly we need a translation service that converts these capabilities into a valid YANG model. Finally, we also need a component that receives configuration commands via NETCONF and turns them into the required domain-specific messages so that the devices can be managed using standard mechanisms.

Ideally there should exist a common protocol or language that let the managed devices express their capabilities in a domain independent manner and that is translatable to a YANG model understood by NETCONF management solutions. Because such a common language is currently not available we designed a simple JSON message format as a minimal working example.

The NETCONF/MQTT bridge is implemented using several Python libraries. The bridge acts as an MQTT client towards the IoT domain, that is both publishing and subscribing to certain topics. The MQTT functionality is provided by the `paho-mqtt` library[3]. The `netconf` server library[4] provides functionality for the NETCONF server part of the NETCONF/MQTT bridge. YANG models are dynamically generated using the `pyang` library[5].

The functionality of the NETCONF client has been tested using the `ncclient` Python library[6], whereas the MQTT broker uses `mosquitto`[7].

[3]http://www.eclipse.org/paho/clients/python/docs/
[4]https://github.com/choppsv1/netconf
[5]https://github.com/mbj4668/pyang
[6]https://github.com/ncclient/ncclient
[7]https://mosquitto.org

4.1 Architecture

Our network architecture is shown in Figure 2 and consists of two separate network management domains. In each domain network and device management tasks are carried out using the respective protocol. This is a purely conceptional separation chosen to illustrate the use case.

However, enforcing a strict separation between the domains may have some additional benefits that we would like to mention briefly. The NETCONF/MQTT bridge could serve as a demarcation point that limits visibility and reachability of the constrained network from public networks. This configuration makes it easy to implement application level checks and security methods at the bridging point.

A defined service entry point also makes it easier to implement rate limiting and user authentication. It also limits the exposure to Denial-of-Service attacks against constrained network devices, that may otherwise be trivial to carry out. It also aligns well with the provisioning of managed services, where service providers deploy and configure residential and customer gateways.

4.2 MQTT topic levels

MQTT can, amongst other things, be used for configuration and device management tasks. However, the protocol makes no dedicated provisions for such use. Topics and message formats have to be coordinated between all communicating entities in the domain, making multi-vendor support challenging. Topics have no directionality, which means, that the protocol does not distinguish between outgoing or incoming messages. All MQTT client entities are free to register as a subscriber or publisher of data and can do so simultaneously for different topics. In order to restrict access to configuration data to trusted entities, additional authorization measures would have to be implemented at the MQTT broker.

For the purpose of the experiment, we defined and used the following topics for device management within the MQTT domain:

```
yang/config
command/led/UUID#
```

The topic `yang/config` will be used to send device configuration data to the NETCONF/MQTT bridge. Whereas

```
{
  "device": {
    "description":"MQTT-Device identified by UUID",
    "uuid": {
      "type":"string",
      "value":"F97DF79-8A12-4F4F-8F69-6B8F3C2E78DD"
    },
    "device-category": {
      "description":"Identifies the device category",
      "type":"string",
      "value":"LED-LAMP"
    }
  },
  "rpc": {
    "set_color_green": {
      "description":"Set the LED-bulb color to green",
      "mqtt-command":"GREEN",
      "input": {
        "uuid": {
          "description":"Sends request to UUID",
          "type":"string"
        }
      }
    }
  }
}
```

Listing 1: JSON encoded configuration and state data.

the topic `command/led/UUID#` is used as a command channel from the bridge to the corresponding device identified by a specific UUID.

4.3 Device Configuration Discovery

The NETCONF/MQTT bridge needs to discover the functionality and configuration of attached IoT devices. We devised a simple mechanism where the bridge receives configuration data about attached devices via a specific MQTT topic. The bridge subscribes to `yang/config` where it listens for incoming configuration messages.

Configuration data is currently modelled in JSON. The format is chosen to closely match the YANG model that will be generated from it. Listing 1 and 2 show an example of such a configuration message and the YANG model that is derived from it. Besides device configuration functions, the JSON message also contains data items that represent operational state for individual devices, such as its UUID value. This data is later used by the bridge to accurately announce the device state within NETCONF `<get/>` RPC calls. Listing 3 and 4 show such a message exchange between the NETCONF client und the bridge. The bridge announces a list of two active devices of a certain type, distinguished by their UUID value. This announcement is generated from the JSON messages received via MQTT.

NETCONF distinguishes clearly between operational and configurational state of a device. We currently only model the operational state, since managing devices via proprietary RPC function calls reflects the current state of the IoT domain management. Therefore the bridge has no device-independent data-store that could be managed. However, if we assume that a common IoT device management standard will emerge in the future, we could very well also manage configurational state at the NETCONF bridge.

4.4 NETCONF operation

A NETCONF client needs to have complete knowledge of device functions and manageable parameters of the managed device. The standard defines the use of static YANG data models, which may have different, clearly referenced revisions. YANG schemas may be provided directly by the NETCONF server and can be discovered from the client via a `<get-schema/>` RPC described in [9]. However, most use cases assume that the data models are exchanged outside an active NETCONF session.

```
<?xml version="1.0" encoding="UTF-8"?>
 <rpc-reply xmlns="urn:ietf:params:xml:ns:netconf:
↪  base:1.0"
 message-id="2">
 <data xmlns="urn:ietf:params:xml:ns:yang:
↪  ietf-netconf-monitoring">
 <![CDATA[module mqtt-led {
  namespace "http://ipv6lab.beuth-hochschule.de/led";
  prefix led;

  container device {
    description "MQTT-Device identified by UUID";
    list device-id {
      key "uuid";
      leaf uuid {
        type string;
      }
    }
    leaf device-category {
      description "Identifies the device category";
      type string;
    }
  }
  rpc set_color_green {
    description "Set the LED-bulb Color to green";
    input {
      leaf uuid {
        description "Sends request to UUID";
        type string;
      }
    }
  }
}
]]></data>
</rpc-reply>
```

Listing 2: Dynamically generated YANG schema.

This paper deviates from the standard and defines a new type of YANG model. The important feature of this model is that is dynamic. It is generated to represent a snapshot of the underlying IoT device state. While it aims to be a true representation of the current network, it is in itself not reproducible in time and therefore not referenceable through the existing versioning mechanism.

We use a dynamically generated YANG model as a virtual repository, which represents capabilities of the managed devices. It only exists on the bridge and needs to be discoverable by the NETCONF clients. For this purpose, the generated YANG model can be retrieved directly from the NETCONF server represented by the bridge.

The dynamic YANG model has no meaning outside the local setting and will be acquired by a NETCONF client using the `<get-schema/>` RPC. In order to inform a connected NETCONF client that dynamic changes have happened to the YANG model, we propose the use of *Event Notifications* defined in [3] and [1].

Our prototypical implementation of a NETCONF/MQTT bridge uses JSON encoded device configurations, received via MQTT (cf. Step 1 and 2 in Figure 2). It generates a corresponding YANG module for each device category. List-

```
<rpc message-id="1"
↪  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0">
  <get/>
</rpc>
```

Listing 3: NETCONF `<get/>` request.

ing 2 shows the generated YANG model, which is derived from the device configuration (Step 3). The NETCONF client now has the ability to discover this model from the NETCONF/MQTT bridge (Step 4). In the next step the NETCONF client uses this YANG module to manage devices in the IoT domain via standard NETCONF methods (Steps 5, 6 and 7).

We chose to model each device category as a separate YANG module. This makes it easier to group device state and management functions together and reduces the rate of change, because we expect individual device functions to be quite stable over time.

Since we keep no configurational state a the NETCONF/MQTT bridge, we represent device functions through custom RPCs within the device model. Such RPCs map to a set of actions or commands on a dedicated device. A decision had to be made at this point on how to model the command. We could either provide a generic function, such as `set_color()` that needs to be parameterized with the correct color value or provide a specific function particular to the color supported by device, such as `set_color_green()`. We chose the latter approach since it also makes the device more discoverable. The decision may ultimately depend on the number of offered choices. If the number of choices is high, it makes more sense to provide a parameterized function.

This type of decision is typical for any interface design and is usually solved by an interface design guideline. Listing 5 shows an exemplary RPC targeted at one particular device in the IoT network. The UUID input parameter is used by the bridge to specify the MQTT topic (or address) for the message generated by the bridge (cf. Section 4.2).

The `mqtt-command` entry in the JSON model (cf. Listing 1) provides the mapping between the NETCONF RPC and the corresponding MQTT message content. The MQTT message content is later interpreted as a command value by the controlled device.

Throughout this paper we assume a simple trust relationship between the managed devices, the NETCONF/MQTT bridge and the NETCONF client. This may not be realistic for practical installations, but a deeper discussion is out of scope for this paper.

5. DISCUSSION

The current development towards an Internet of connected devices is handicapped by the lack of a common configuration and management approach. This creates disjoint sets of devices that lack true interoperability, thereby missing new service opportunities, such as automated deployment and managed IoT-services for devices manufactured by different vendor groups.

This paper examines an exemplary solution for IoT devices management based on standards. We map our proprietary, MQTT-based device management to dynamically generated NETCONF RPCs that may be discovered via a corresponding YANG model. In our approach, IoT devices

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc-reply
↪  xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
  message-id="1">
  <data>
    <device
↪  xmlns="http://ipv6lab.beuth-hochschule.de/led">
      <device-id>
       <uuid>F97DF79-8A12-4F4F-8F69-6B8F3C2E78DD</uuid>
      </device-id>
      <device-id>
       <uuid>F97DF79-8A12-4F4F-8F69-6B8F3C2E88FF</uuid>
      </device-id>
      <device-category>LED-LAMP</device-category>
    </device>
  </data>
</rpc-reply>
```

Listing 4: Generated NETCONF `<get/>` response.

```
<?xml version="1.0" encoding="UTF-8"?>
<rpc xmlns="urn:ietf:params:xml:ns:netconf:base:1.0"
↪  message-id="2">
  <set_color_green
↪  xmlns="http://ipv6lab.beuth-hochschule.de/led">
    <uuid>F97DF79-8A12-4F4F-8F69-6B8F3C2E78DD</uuid>
  </set_color_green>
</rpc>
```

Listing 5: RPC from the NETCONF client containing UUID.

post their configuration data via a previously agreed MQTT channel and allow a NETCONF/MQTT bridge to discover, learn and propagate their capabilities. Instead of creating yet another network management protocol we can show that it is possible to reuse existing approaches in order to reduce parallel development and avoid marginalisation.

5.1 Configuration

One very important building block for a truly standardized way for device configuration, is the agreement on a data format that describes IoT resources, entities and services, that is translateable into YANG models. Developing device ontologies is an ongoing research and standardization item for organisations such as oneM2M[8] and W3C[9], however no final consensus has been reached yet and active deployment seems to be marginal. We are currently investigating if it will be possible to transform our custom JSON data model into a oneM2M ontology, thus adding another piece of standardisation to the scenario.

Another important aspect, that we could only touch briefly is IoT device lifetime management. This includes discovery of the current status of the device, its network association and other parameters. We currently implement a very limited life-time model, where a device may post its recent configuration as part of an *on-boarding* procedure as soon as it detects a network connection or reset. Large pieces for a full data-model based IoT device lifecycle management are still missing and should be a topic for future investigations. For example, we have not yet implemented a mechanism that makes sure that IoT device data represents the actual status of the device. Configuration data may be out of sync with the deployed device.

[8]http://www.onem2m.org
[9]http://www.w3.org/Submission/2015/SUBM-iot-lite-20151126/

5.2 MQTT usage

MQTT is easy to work with and is a protocol with interesting properties. It is, however, focussing on message transport and provides little coordination at the topic level or application layer abstractions for data modelling. It is currently the responsibility of each application developer to have an intrinsic knowledge of the final deployment and the currently used topic separation may clash in larger installations. Hence MQTT should be understood only as pure transport protocol, implementing a publish/subscribe architecture. Additional functionalities as given by supporting e.g. a NETCONF/YANG protocol suite are currently left to the application developer.

5.3 Ephemeral YANG models

Throughout this paper we are using YANG models outside the current specification. RFC 6020 defines YANG models to be static and referenceable. We generate ephemeral models that may be short lived and change very frequently. The generated models capture the current capabilities of the connected devices and have no defined meaning outside the context of the currently managed IoT domain.

Apart from this change in usage of YANG models, no other changes have been made to the underlying protocols. If we assume that a universal ontology for IoT emerges in the future and that it would be fully translatable into YANG, our approach might no longer be needed and device models could again be statically described. However, in the meantime we see our approach as a way towards a workable IoT device management based on standard mechanisms and a viable way to integrate legacy device installations.

5.4 Influence on Protocol Design

It has been our experience that IoT device management is very fragmented and most network management approaches tend to be device oriented. The Netconf standardization effort is a step in the right direction as it uses a discoverable data model that can be analyzed and processed. However, data and device aggregation is usually done at the Netconf Client in post-processing, rather than at the Netconf Server. This potentially generates traffic and processing requirements that are difficult to meet in an IoT setting based on restricted devices. We believe that it would be beneficial to treat YANG models not exclusively as normative references but to support derived models that may be preprocessed or dynamically generated at the Netconf Server.

The current approach for YANG data modelling is still difficult to automatize and keeps humans in the loop. As YANG models try to bridge the gap between device ontologies and configuration repositories, we see value in developing them in a direction that allow rich device descriptions to make automatic device discovery and configuration possible.

6. CONCLUSIONS

It has been our aim to extend standards-based configuration management to IoT installations. We implemented the concepts and ideas discussed in this paper in a working prototype and where able to validate the operational value of such an approach. The corresponding code can be found on Github[10].

[10]https://github.com/tscheffl/netconf-mqtt-bridge

The format and mechanism for configuring the IoT domain is currently heavily inspired by the properties of the YANG data modelling language and the available NETCONF mechanisms. We are further investigating opportunities and consequences of an alternative approach for the generation of device descriptions based on ontologies.

7. ACKNOWLEDGMENTS

We would like to thank Jürgen Schönwälder of Jacobs University in Bremen for his valuable insight into the current and past activities and decisions of the relevant IETF working groups.

8. REFERENCES

[1] A. Bierman. Network Configuration Protocol (NETCONF) Base Notifications. RFC 6470, Internet Engineering Task Force, Feb. 2012.

[2] M. Bjorklund. YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020, Internet Engineering Task Force, Oct. 2010.

[3] S. Chisholm and H. Trevino. NETCONF Event Notifications. RFC 5277, Internet Engineering Task Force, July 2008.

[4] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman. Network Configuration Protocol (NETCONF). RFC 6241, Internet Engineering Task Force, June 2011.

[5] R. T. Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. PhD thesis, University of California, Irvine, 2000.

[6] K. Hartke. Observing Resources in the Constrained Application Protocol (CoAP). RFC 7641, Internet Engineering Task Force, Sept. 2015.

[7] Oasis Standard - MQTT Version 3.1.1. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf, Oct. 2014.

[8] V. Perelman, J. Schoenwaelder, M. Ersue, and K. Watsen. Network Configuration Protocol Light (NETCONF Light). Internet-draft (expired), Internet Engineering Task Force, Jan. 2012.

[9] M. Scott and M. Bjorklund. YANG Module for NETCONF Monitoring. RFC 6022, Internet Engineering Task Force, Oct. 2010.

[10] A. Sehgal, V. Perelman, S. Kuryla, and J. Schönwälder. Management of resource constrained devices in the Internet of Things. *IEEE Communications Magazine*, 50(12):144 –149, Dec. 2012.

[11] Z. Shelby, K. Hartke, and C. Bormann. Constrained Application Protocol (CoAP). RFC 7252, Internet Engineering Task Force, June 2014.

[12] P. van der Stok, A. Bierman, M. Veillette, and A. Pelov. CoAP Management Interface. Internet-draft, Internet Engineering Task Force, Jan. 2017.

[13] M. Veillette, A. Pelov, A. Somaraju, R. Turner, and A. Minaburo. CBOR Encoding of Data Modeled with YANG. Internet-draft, Internet Engineering Task Force, Feb. 2017.