This is an electronic reprint of the original article.
This reprint may differ from the original in pagination and typographic detail.

Novo Diaz, Oscar

Making Constrained Things Reachable : A secure IP-agnostic NAT traversal approach for IoT

Published in:
ACM Transactions on Internet Technology

DOI:
[10.1145/3230640](https://doi.org/10.1145/3230640)

Published: 01/10/2018

Document Version
Peer reviewed version

Please cite the original version:
Novo Diaz, O. (2018). Making Constrained Things Reachable : A secure IP-agnostic NAT traversal approach for IoT. *ACM Transactions on Internet Technology*, 19(1), [3]. <https://doi.org/10.1145/3230640>

This material is protected by copyright and other intellectual property rights, and duplication or sale of all or part of any of the repository collections is not permitted, except that material may be duplicated by you for your research use or educational purposes in electronic or print form. You must obtain permission for any other use. Electronic or print copies may not be offered, whether for sale or otherwise to anyone who is not an authorised user.

Making Constrained Things Reachable: A Secure IP-agnostic NAT Traversal Approach for IoT

OSCAR NOVO, Ericsson Research, Finland

The widespread adoption of the Internet of Things (IoT) has created a demand for ubiquitous connectivity of IoT devices into the Internet. While end-to-end connectivity for IoT requires in practice IPv6, a vast majority of nodes in Internet are only IPv4-capable. To address this issue, the use of Network Address Translation (NAT) at the IoT network boundary becomes necessary. However, the constrained nature of the IoT devices hinders the integration of traditional NAT traversal architectures through IoT networks.

In this paper, we introduce a novel transition mechanism that transparently enables IoT devices behind NATs to connect across different network-layer infrastructures. Our mechanism adopts the IoT standards to provide a global connectivity solution in a transparent, secure, and elegant way. Additionally, we revisit the NAT solutions for IoT and describe and evaluate our current implementation.

CCS Concepts: • **Networks** → **Naming and addressing**; *Transport protocols*;

Additional Key Words and Phrases: M2M, IoT, Constrained Networks, NAT traversal

ACM Reference Format:

Oscar Novo. 2017. Making Constrained Things Reachable: A Secure IP-agnostic NAT Traversal Approach for IoT. *ACM Trans. Internet Technol.* 1, 1, Article 1 (January 2017), 20 pages. <https://doi.org/0000001.0000001>

1 INTRODUCTION

In recent years, several industries have innovated new technologies and applications motivated by the Internet of Things paradigm. These new technologies and IoT applications are expected to bring many benefits to consumers and industries, serving different domains such as health care, home automation, or transport. On the other hand, constrained physical IoT devices must be addressable and identifiable in a global network infrastructure. In other words, IoT technologies depend upon connectivity in order to exchange information among constrained physical devices embedded with transducers (sensors and actuators).

With a predicted 18 billion devices by 2022[10] and the depletion of all the IPv4 addresses, the adoption of IPv6[8] –with its 340 undecillion IP addresses– will be the only possible path to the success of IoT. Although IPv6 was introduced long time ago, the deployment of the Internet Protocol Version 6 (IPv6) has been relatively slow in the Internet, hence the majority of the networks are still IPv4-based. However, a considerable number of IoT technologies specially for low power devices are designed for only IPv6. This includes, for example, 6LoWPAN[14] which supports IPv6 over IEEE 802.15.4[12], ZigBee[28] short-range communication technologies, or the IPv6-based networking protocol called Thread[24] aimed at home automation environments.

Author's addresses: Oscar Novo, NomadicLab, Ericsson Research, Finland;

Author's address: Oscar Novo, Ericsson Research, Hirsalantie 11, Jorvas, 02420, Finland.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2018 Association for Computing Machinery.

1533-5399/2017/1-ART1 \$15.00

<https://doi.org/0000001.0000001>

Due to the lack of support of IPv6, some IoT networks will have to interact, in the majority of the cases, with IPv4-based networks on the Internet. Even though a wide range of methods to deal with IPv4 interconnectivity exist, such as IPv6 over IPv4 tunnelling and network address translation (NAT) mechanisms, these transition mechanisms will need to evolve beyond the conventional Internet scenarios to support specific IoT scenarios.

IoT technologies are particularly optimized for low bandwidth network environments, low power consumption and low computing power, making the adoption of existing IP transition mechanisms challenging due to their impact in the throughput and CPU utilisation. In addition, the IoT paradigm differs from the conventional Internet scenarios. Consequently, the existing IP transition mechanisms are not suitable for supporting IoT. New mechanisms are expected to take into consideration all the different IoT constraints and adapt them efficiently into the new IoT paradigm.

In this paper, we design and implement an efficient IP network transition method for CoAP devices[21] based on network address translation. Our solution modifies an IoT entity called *Resource Directory*[22] in order to manage through NATs the incoming IPv4 requests of the constrained networks. Additionally, the proposed method allows to seamlessly discover the resources in a constrained network environment through the resource directory and enables establishing a connection to them. Thus, the result of our implementation supports communication between IPv6-based constrained network environments and IPv4-based network.

The rest of this paper is organized as follows. In Section 2, we introduce the terminology and current NAT technologies and their impact in IoT. In Section 3, we motivate our work by analysing the limitations of the current mechanisms and, in Section 4, we describe our design. The implementation of our proposed IoT architecture is described in Section 5 and the results of our performance study in Section 6. The conclusions and future work is presented in Section 7.

2 BACKGROUND

This section reviews the NAT terminology and taxonomy adopted by this paper, as well as provides an overview of the existing NAT traversal solutions, starting from the established approaches and proceeding to the latest proposals. The section concludes with a brief description of the challenges in adopting the current NAT traversal approaches in IoT scenarios.

2.1 Taxonomy of NATs

In discussing the architectural impact of NAT in the IoT, the first task is describing the classification of NATs, since NATs do not behave uniformly. This section only covers terms that are required for the understanding of this paper. A detailed discussion and further information is given in [2, 20, 23].

The most basic classification of NATs (in order of complexity) is static NAT, dynamic NAT and Network Address Port Translation (NAPT) NATs. While static NATs maintain a static association between IP addresses, dynamic NATs assign dynamically IP addresses from a predefined pool of addresses when an internal host needs access to the Internet. On the other hand, NAPT uses a single external IP address and maps each internal host with a different port of that address. Responses from the Internet include the originating port so that the router knows which internal IP address to map the response to. Although the NAPT mechanism[26] has been deprecated[1], the solution is currently in use since there are still scenarios in which a revised version of the standard can be a suitable solution.

The *Simple traversal of UDP through NATs* (STUN) [20] standard classified the NAT implementation as how UDP bindings are managed within a NAT. The standard has traditionally classified the NAT implementation into four types of behaviours: full-cone NAT, restricted-cone NAT, port-restricted cone NAT and symmetric NAT. However, this document and its classification was

NAT Behaviour		
Mapping	Port Mapping	Port Preservation Port Overloading Port Multiplexing
	NAT Mapping	Endpoint Independent Endpoint Address Dependent Endpoint Address and Port Dependent
	Timer Refresh Mapping	Bidirectional Outbound Inbound Transport Protocol State
Filtering	Endpoint Filtering	Endpoint Independent Endpoint Address Dependent Endpoint Address and Port Dependent
	Timer Refresh Filtering	Bidirectional Outbound Inbound
Hairpinning		

Table 1. NAT Classification

deprecated[19] as this classification was inadequate and too simplistic to correctly described many NATs. The document was replaced by an approach that tries to classify them by their behaviour. That terminology was described in the *Network Address Translation (NAT) Behavioural Requirements for Unicast UDP*[2] document and describe two behaviours: mapping and filtering. Mapping relates to the strategy used by the NAT to assign a public address to a new state in the NAT. Filtering relates to the NAT handling the packets trying to use an existing mapping.

Table 1 shows the different categories. The *Port Mapping* is divided into three cases: in the *Port Preservation* case the NAT attempts to preserve the local port number. The new connection takes over the old connection in the *Port Overloading* and the *Port multiplexing* category multiplex the ports based on the source transport address. In this particular behaviour, the incoming packets can carry the same destination port.

NAT Mapping deals with reusing the port binding for subsequent NAT sessions initiated from the same internal IP address. In the case of the *Endpoint Independent* category, the port is used for any external IP address and port. The *Endpoint Address Dependent* only reuse the same mapping for the same external IP address and the *Endpoint Address and Port Dependent* category reuses the same mapping for the same external IP address and port.

The *Timer Refresh Mapping* classifies the NATs according to how it resets the timer in the bindings. While *Bidirectional* category refreshes the timer every time a packet goes through the NAT, the *Outbound* only resets the timer for the outgoing packets and the *Inbound* only from the incoming packets. The *Transport Protocol State* links the timer to the transport protocol of the mapping, for instance to the keepalive time of the TCP connections.

Endpoint Filtering describes how existing NAT mappings are handling incoming connections. In the *Endpoint Independent* category the NAT does not filter the outgoing packets of a mapping. The *Endpoint Address Dependent* filters the outgoing packets according to the source address and the *Endpoint Address and Port Dependent* filters outgoing packets according to the packets and ports. The *Timer Refresh Filtering* is similar to the bidirectional, outbound and inbound cases explained in the *Mapping* category.

The *Network Address Translation (NAT) Behavioural Requirements for Unicast UDP*[2] document also has another type of classification according to their hairpinning behaviour. Hairpinning allows

two devices on the internal side of the NAT to communicate even if they only use each other's external IP addresses and ports.

2.2 Current NAT Traversal Solutions

Due to the complexity of NATs, devices behind NATs have to come up with different approaches in order to have stable outbound connections with other device behind NATs and to be reachable.

One of the first attempts to solve the NAT connection problem was STUN. STUN[19] is a mechanism for devices inside a NAT that enable to query their public mapped address. In addition to that, STUN used to classify the NAT behaviour[20] but the use of NAT type detection was deprecated in STUN since the behaviour of NAT varies overtime. STUN is extremely lightweight and scalable but it has the disadvantage of not working through certain types of NATs limiting its applicability. TURN tried to solve this problem. TURN[16] is a method to rely messages between two devices behind a NAT through a third-party server. Even though TURN works in more cases than STUN does, relaying is expensive and creates latency. The Interactive Connectivity Establishment (ICE) [18] combined both TURN and STUN methods to find a better solution for NAT traversal. ICE defines a technique to discover a network path between devices behind NATs using TURN and STUN.

UDP hole punching is another well-established technique in which a device behind a NAT *punches* a hole in the NAT allowing an external device to successfully reach inside the NAT. This simple technique does not really work when both devices are behind a NAT and is more commonly use to maintain a NAT connection open since the majority of NATs close a connection after a period of inactivity.

On the other hand, many peer-to-peer applications need a way to instruct a NAT device to open a port to access the devices in their networks. That technique is called *port forwarding* and specifications like UPnP have implemented protocols such as the Internet Gateway Device Protocol (IGD)[13] to automatically configure *port forwarding* in NAT nodes. A UPnP-enabled application client can determine if a NAT device is present, learn the translated IP address and configure port mappings in the IGD. Other protocols implementing *port forwarding* are the *NAT Port Mapping Protocol* (NAT-PMP)[7] implemented by Apple as an alternative of IGP. NAT-PMP was designed for residential NAT gateways. Hence, the limitations of NAT-PMP such as the support of a single external IP address induce the creation of the *Port Control Protocol* (PCP)[27]. PCP is the successor of NAT-PMP and its applicability is broaden than NAT-PMP supporting multiple external IP address. Nonetheless, PCP shares similar semantics and concepts than NAT-PMP.

Application-Level Gateways (ALG) also supports the functionality of NAT traversal. *Application-Level Gateways* intercept traffic of specific applications and change it to traverse the NATs. Nowadays many NAT devices are equipped with intelligence logic to modify specific traffic (such as VoIP or online gaming traffic) in its effort to help it with the NAT traversal.

In addition to all these methods and specifications, there are a set of IPv6 transition mechanism used to facilitate the transitioning of IPv4 infrastructures into IPv6. *IPv6 Rapid Deployment*(6rd)[9] is one of these methods and 6to4[6] is another. Even though that 6to4 is not widely used since some parts have already been deprecated[25].

Furthermore, the Domain Name System must be updated to deal with IPv6. There are two types of mechanisms for DNS translation: DNS64 and DNS46. DNS46 is a transition mechanism that synthesizes IPv6 records into IPv4 records. On the other hand, DNS64[4] synthesizes IPv4 records into IPv6 records. However, DNS46 was considered harmful and it was deprecated by the IETF.

2.3 NAT Traversal in IoT

The requirements of the of low-power, low-cost IoT devices has resulted in the design of specific protocols that address these needs. MQTT[17] and the *Constrained Application Protocol* (CoAP)[21] are lightweight application layer protocols designed for the communication of constrained devices. While MQTT was designed as a communication protocol in the publish/subscribe architecture, CoAP was designed as a protocol in the REST[11] architecture. Whereas REST is transaction oriented (single operations between devices), publish/subscribe is data oriented (devices subscribe to data) and requires a central MQTT broker to manage and route messages among an MQTT network.

Moreover, the IPv4 address space has already been depleted and IPv6 is the only Internet Layer protocol that can accommodate all the IoT devices. However, it will be often necessary for the IoT devices to interact with IPv4 nodes or across IPv4-only networks. There are several ways to deal with IPv4 interconnectivity, with *address translation* being the most popular mechanism.

MQTT and CoAP have fundamental differences which makes them behave differently through NATs. In the IoT publish/subscribe model, MQTT clients establish an outgoing TCP connection to a MQTT broker to fetch the information from it. Typically, the MQTT brokers are not located behind NATs and the reception of packets over an established TCP connection do not create problems for devices behind NATs.

On the other hand, CoAP devices generally act as CoAP servers providing resources which can be accessed by other nodes. In that case, CoAP devices behind a NAT must be able to receive inbound packets. However, NATs will prevent such inbound communication. Currently, CoAP devices behind a NAT must first initiate the communication with external nodes allowing the NAT to associate them both or use *hole punching* techniques to open ports in the NAT. Another alternative is defined in LWM2M[15] where LWM2M clients behind a NAT can use the *Queued Mode*[15]. The *OMA Lightweight M2M* is a protocol from the *Open Mobile Alliance* for IoT device management. In the *Queued Mode*, LWM2M servers keep the request of the LWM2M clients. The LWM2M clients, which are behind the NAT, use the *polling* technique to access the information in the LWM2M Servers.

Despite the fact the *Queued Mode* provides a solution for LWM2M devices behind a NAT, it uses *polling* which introduces latency up to the polling interval as well as consumes unnecessary energy on the endpoints making it infeasible for other IoT scenarios.

The mentioned solutions do not allow external CoAP devices to initiate a communication with internal nodes behind a NAT which is a hurdle for the adoption of IoT. In addition to that, CoAP is based on the exchange of messages over UDP and the connections initiated by the CoAP clients will typically time-out after a minute in the majority of NATs[2]. Furthermore, the constrained nature of the IoT devices make it inefficient to keep a connection open using *UDP hole punching* techniques. In addition, solutions like the LWM2M *Queued Mode* approaches are too specific for management and do not suit some of the IoT scenarios such as real-time IoT scenarios.

3 MOTIVATION

IoT encompasses a broad range of devices with highly heterogeneous capabilities. Many of them are characterized by high memory, battery and CPU boundaries being capable of handling a large number of tasks but a lot more have low capacity. According to [5], IoT devices can be classified in terms of their constraints. The most constrained devices are *Class 0* devices. Class 0 devices are very limited in memory and processing capabilities. Those devices have to rely on proxies and gateways to participate in Internet communications. Even though *Class 0* devices are very limited, they still can act as CoAP servers and, according to [5], can provide simple operations such as

	LW	CoAP Compliant	Transparent	Secure
Session Traversal Utilities for NAT (STUN)	Yes	No	Use STUN Server	Yes
Traversal Using Relays around NAT (TURN)	No	No	Use Relays	Yes
Interactive Connectivity Establishment (ICE)	No	No	ICE Negotiation	Yes
IPv6 Rapid Deployment (6rd)	Yes	NA	IPv4 Tunnelling	Unsolicited Outbound Network Traffic
Hole Punching	No	No	UPD Keep-alive	Yes
Universal Plug and Play (UPnP) (IGD, NAT-PMP, PCP)	No	No	Instruct NAT Device	Yes
Application-Level Gateway (ALG)	Yes	No	Yes	No End-to-End Security

Table 2. Gap Analysis of the Existing NAT Solutions

controlling an electrical switch. On the other hand, *Class 1* devices are still quite constrained in memory and processing capabilities. However, they are capable to use a constrained protocol such CoAP over UDP. *Class 2* are the least constrained devices. They are capable of supporting most of the protocol stack and have higher capabilities.

This paper envisions to provide a secure IPv4/IPv6 translation solution for most of the *class 0* and *1* CoAP devices. The solution focus on NAT techniques since other solutions such *IP dual stack* are infeasible for *class 0* and *1* devices. Besides that, NAT techniques prevent unsolicited outbound network traffic to reach the NATed network, protecting the devices inside the NATed network.

Even though there is a large number of existing NAT solutions, the majority of techniques are resource consuming and are not appropriate for *class 0* and *1* devices. As a result, the current NAT behaviour and the traditional NAT scenarios prevent the proliferation of some IoT scenarios. According to our understanding, a NAT solution should have the following characteristics to fulfil the basic requirements of such devices:

- *Lightweight*: *Class 0* and *1* devices have a limited memory and CPU capabilities. In addition, such devices reside in constrained networks and are only capable of hosting constrained protocol stacks. It is advisable to provide NAT solutions where complexity, as well as computation time and memory overhead is reduced as much as possible. In other words, the solution should include NAT mechanisms design to interact as little as possible with the IoT devices.
- *CoAP compliant*: The solution proposed in this paper aims for constrained devices which support the Constrained Application Protocol (CoAP). That, in turn, leads to the NAT solutions to comply with the IoT standards; not only being able to understand CoAP but to be able to integrate the new IoT components into the NAT solutions.
- *Transparent*: Due to the very limited capabilities of the IoT devices and its limited energy supply, transparency is considered to be especially desirable in this case. Furthermore, preventing the devices to be aware of the network, increase their compatibility and interoperability in the IoT scenarios.
- *Secure*: The solution should allow end-to-end security between devices. In addition, it is desirable that unsolicited outbound network traffic do not reach the devices inside the NATed network.

Table 2 summarizes the shortcomings of the current NAT traversal methods described in 2.2. While *STUN* is a lightweight method, it is not CoAP-compliant and clients need to be NAT aware.

Fig. 1. Logical Architecture of the NAT Implementation

TURN relays the traffic through *TURN* servers which are costly in terms of latency and bandwidth. *ICE* combines *TURN* relays in its solution. Hence, *ICE* has the same drawbacks than *TURN* when is used in *ICE*. Additionally, *ICE* negotiations are not CoAP-compliant.

6rd is a tunnelling technology that allows IPv6 devices to communicate with each other over IPv4 networks. Therefore, *6rd* does not provide a solution when devices belong to networks with different IP versions.

Furthermore, the *hole punching* technique is impractical and resource consuming for IoT devices. On the other hand, the drawback of *UPnP* is its lack of transparency since clients are aware of the NAT devices and control them using specific protocols. In addition to the foregoing techniques, *ALG* - to the best of our knowledge - does not have any extensions for IoT protocols and is impractical for end-to-end security.

Our approach provides a solution to overcome all these limitations in the current NAT solutions. As a result, our solution enables outbound connections and is totally transparent to the devices, optimizing the utilization of the resources in the constrained devices. In addition, the solution do not modify the IoT standards which makes it even easier to integrate with the existing IoT scenarios.

4 PROPOSED ARCHITECTURE

In order to enable ubiquitous connectivity between CoAP devices located in different IP networks, we have developed and implemented a high-level network architecture, which is depicted in Figure 1. The main target is the integration and seamless communication between CoAP devices with different network-layer infrastructure.

4.1 Components

On a high level, the network is divided into logical entities with the following responsibilities:

- *Gateways* regulate the traffic between two dissimilar networks. A gateway acts as translating proxy and is a necessary piece in constrained networks that want to interact with networks with different network-layer infrastructures.
- *CoAP Resource Directory (RD)*[22] provides a mechanism to discover IoT resources. The resource directory is an IoT entity used as a repository to keep track of the IoT resources of the constrained networks. It implements REST interfaces for discovery, registration, and

lookup of those resources. Having the resource directory located in the CoAP gateway gives the possibility for the nodes in both networks to discover the different resources in the constrained network.

In our solution, a new NAT logic has been implemented in the resource directory which allows to dynamically monitor, store, and manage NAT rules through a PCP client.

- *Domain Name System64 (DNS64)* is – according to [4]– a mechanism for synthesizing AAAA records from A records. A and AAAA records are primary DNS records that associate a domain name with a specific IP address. In our implementation, DNS64 is used with an IPv6/IPv4 translator to enable client-server communication between IPv6-only and IPv4-only nodes, without requiring any changes to either the IPv6 or the IPv4 nodes, for the class of applications that work through NATs.

- *Network Address Translation from IPv6 Clients to IPv4 Servers (NAT64)*[3] is a mechanism for IPv4-IPv6 transition and IPv4-IPv6 coexistence. Together with DNS64, these two mechanisms allow an IPv6-only client to initiate communications with an IPv4-only server.

Our implementation uses a stateless NAT64. It simply performs a one-to-one substitution of IP addresses using a mapping table provided by the network administrator. The mapping table defined by us allocates a set of addresses from the private IPv4 address space.

- *Network Address Translation (NAT44)*[26] is a mechanism for IPv4-IPv4 translation. Typically, the translation is done between a private IPv4 address and a public IPv4 address.

Since CoAP runs over UDP, our NAT implementation does not need to have port preservation. Indeed, multiple UDP communications can reuse the same UDP port to send packets to different nodes. That reduces significantly the number of address translations used by the resource directory making the design of our NAT44 easier, as only a single external IPv4 address is required to map all the internal communications.

- *Port Control Protocol (PCP)*[27] is a mechanism that allows a node to control how incoming IPv4 packets are translated and forwarded by a NAT and also allows a host to optimize its outgoing NAT keepalive messages.

The PCP server in our implementation is deployed unmodified inside the CoAP gateway to restrict external access in such a way that only nodes in the CoAP gateway can access the server. Thus, we ensure that external nodes cannot modify the rules in the NAT.

- *Endpoints* are constrained devices connected via a short-range technology, such as Wi-Fi or IEEE 802.15.4[12]. Depending on the device type, the devices provide a CoAP client or a CoAP server interface for actuation and data access. In some cases, the devices can provide both interfaces altogether.

4.2 NAT Traversal and the Resource Directory

As shown in Figure 1, having the resource directory located in the CoAP gateway gives the possibility for the CoAP endpoints in both networks to discover the different resources in the IPv6-only constrained network.

However, a resource directory cannot itself translate lookups from two dissimilar networks. In that case, if a CoAP endpoint in an IPv4 network intends to communicate with a resource held by an endpoint in an IPv6-only constrained network, the IPv4 endpoint would request the information from the resource directory but the resource directory will return an IPv6 address instead. Supposing that the IPv6 endpoint is behind a NAT64 and the resource directory is in the IPv4 network, the resource directory will return the IPv4 address assigned by the NAT64. However, the associated NAT state will typically time-out after a minute of inactivity since the majority of the NATs refresh their mappings according to the transport protocol state. Methods such as *hole*

Fig. 2. Discovery, Registration and Access of Resource Information

punching with periodic keepalives could be used to keep the connections open but the scarcity of resources in constrained devices along with the sleeping patterns of some IoT devices makes that option infeasible.

The solution we propose modifies the resource directory accordingly to map the IPv6 address of an endpoint into an IPv4 address. In addition, our solution creates static connections in the NAT44 and keeps the connections open during the lifetime of the resources into the resource directory. To achieve this, our solution integrates a *Port Control Protocol (PCP) Client* [27] into the resource directory. Thus, the resource directory keeps track of the resources and creates static rules in the NAT44 using the PCP Client. As a matter of fact, registered resources have soft state in the resource directory and will eventually time-out; the maximum time-out of a resource in the resource directory is one day. After that time, the registration has to be refreshed by the owner of the resource. In our implementation, the resource directory aligns the time-out of the resource with the time-out of the static rules in the NAT44. In case a resource is deleted before its time-out, the resource directory will also delete the static rule from the NAT44.

The scenario depicted in Figure 2 shows all the steps described in this section from the registration of the resources into the resource directory by an IPv6 endpoint until the retrieval of the resource information by an IPv4 endpoints. In this scenario, the resource directory adds a new rule into the NAT44, as well as the NAT64 creates a dynamic IPv6 to IPv4 mapping. Thus, the IPv4 endpoints are able to access the IoT resources inside the IPv6 network.

A CoAP endpoint can hold several resources which means that the resource directory can have several resources with the same IPv4 address belonging to the same CoAP endpoint. Once a resource is deleted from the resource directory, the resource directory has to check if the IPv4 address is used by other resources before deleting the rule from the NAT44. This is one of the reasons whereby we implemented a new logic into the resource directory for tracking the different connections of the resources before modifying the NAT44. In addition, the resource directory stores the lifetime information of every rule and the IPv4 address assigned by the NAT64 to that specific resource.

4.3 Message Processing in the Resource Directory

The schema in Figure 3 explains the logic behind our NAT44 implementation in the resource directory. The logic behind the resource directory relies on three types of CoAP operations sent by the CoAP entities in the IPv6 network: registration (add rule), registration update (refresh timeout) and registration removal (delete rule). According to these three operations, the resource directory selects the best option in the NAT44.

- *Discovery*: Before a CoAP endpoint can make use of a resource directory, it must first know the resource address and port. There can be several mechanisms for discovering a resource directory but in our solution we configure a DNS64 and set up a specific hostname for the resource directory. Therefore, the CoAP endpoints only need the hostname of the resource directory to query the DNS64 and obtain the IPv6 address of the resource directory.
- *Registration*: After discovering the location of a resource directory, a CoAP endpoint may register its resources using the registration interface defined in the resource directory document[22]. The registration accepts a CoAP POST message from an endpoint containing the list of resources to be added to the resource directory as the message payload along with query string parameters indicating endpoint characteristics such as the name of the endpoint, its domain or the lifetime of the registration. The resource directory then creates a new resource entry and returns its location to the CoAP endpoints. The location is just an integer identifier. The CoAP endpoint will use that location to identify the resource in future operations.
- *Registration update*: The registration update operation is used by a CoAP endpoint to refresh or update its registration information in the resource directory. The CoAP endpoint sends a POST request to the resource directory including the location assigned in the registration operation. An update operation might update the lifetime or the context registration parameters of the resource since the last update.
- *Registration removal*: Although the resources in the resource directory have soft state, a CoAP endpoint can explicitly remove resources from the resource directory. This is accomplished by a CoAP endpoint performing a DELETE request and by including the location of the resource assigned by the registration operation in the request.

Once an endpoint registers a new resource, the resource directory assigns a port to that resource and stores that information. If the endpoint has not previously registered a resource, the resource directory will assign a new port to that resource. In addition, the resource directory will store the lifetime of the resource and it will create a new location identifier to identify that resource inside the resource directory.

It should be noted that the ports are assigned in an incremental manner avoiding ports that are being used by other nodes or applications. The maximum number of potential assigned ports available in our solution is 64511. In other words, the solution could assign a different port to almost all the nodes in an IPv6 network with a subnet mask of size 112. The range of the possible assigned ports ranges from 1025 to 65535. Our solution avoids assigning *system* ports to any node.

Typically, upon receiving any request operation, the resource directory will try to update the NAT44 context of that specific resource using the lifetime parameter of that specific request if present. In particular, after receiving a registration request, the resource directory will first look for an available UDP port for the external address. As mentioned before, NATs do not need to have port preservation for UDP connections. Multiple UDP communications can occur on the same source port. Therefore, a single external IPv4 address is enough to map all the internal connections and only an available port is required to map an internal IPv4 address into an external address. If a resource –and consequently a rule in the NAT44– exists in the resource directory, the resource

Fig. 3. NAT logic implemented in the Resource Directory

directory will refresh the time-out of a CoAP endpoint upon requests. First of all, the resource directory will check if there are more resources holding the same internal IPv4 address. In that case, the resource directory will compare the lifetime of those resources with the lifetime of the request and if the lifetime of the request is higher, it will update the NAT rule with the new lifetime accordingly. When performing a registration removal operation or once a resource time-out, the resource directory will first try to look for other resources using the same IPv4 address. Upon success, it will compare the lifetime of the resources and if the lifetime of the deleted resource is the highest among all, it will update the NAT rule with the highest lifetime of the existing resources. Otherwise, it will not update the NAT rule.

Thereafter, a CoAP endpoint outside the CoAP gateway can contact the resource directory to discover all the resources in the IoT network. Once the resources are discovered, the endpoint will use the lookup interface defined by the resource directory[22] to fetch the IPv4 address, port and path of the resource. After receiving the location of the resource – which will be the IPv4 address and port mapped in the NAT44, not the real IPv6 address – the CoAP endpoint will be able to contact the resource. The NAT64 will respectively map the IPv4 address to the IPv6 address of the resource in the IoT network.

4.4 Security Considerations

The goal of our solution is to improve the ability of the resource directory to control the associated NAT state. Controlling a NAT can always result in undesirable security threats. Security in our solution is most properly provided by the fact that the PCP server is running locally on the CoAP gateway. As a result, the NAT can only be controlled by a node in the gateway.

Many NAT solutions do not adequately deal with encrypted data. Our solution, however, accepts end-to-end encryption between the endpoints without hindering the NAT traversal functionality. This comes from the fact that the Resource Directory creates the IPv6 to IPv4 mappings on behalf of the endpoints and the endpoints have a direct communication with the Resource Directory. Once

Fig. 4. Sample Scenario describing the Physical Interfaces, Address Mappings and Interactions between the different Entities of the NAT implementation

the mapping is created, the endpoints can establish a direct end-to-end encrypted communication between each other.

In addition to that, all CoAP communications must be secured by the Datagram Transport Layer Security (DTLS) protocol according to the CoAP [21] standard. DTLS uses encryption to ensure privacy, so that other parties cannot eavesdrop or tamper with the messages being sent.

All mappings that could be created by a specific endpoint behind the NAT using a registration operation are inherently considered to be authorized. By using IP address spoofing, it is possible for an endpoint to register resources on behalf of another node. However, it is not clear how realistic this threat model is, since an endpoint behind a NAT could attack directly the internal endpoints without resorting to such convoluted technique. On the other hand, explicit mappings are linked to the registration operations and the maximum lifetime of a registration is one day.

Even though our solution only expose the CoAP devices that registered to the CoAP gateway, our solution could be improved by adding authentication and authorization to the CoAP gateway to prevent undesired outbound mapping connections.

5 IMPLEMENTATION

To verify our solution, we implemented a prototype. The following section provides additional details about our implementation, in particular regarding the CoAP gateway, CoAP endpoints and the software components.

The *Gateway* is a *Raspberry Pi 2 Model B*, a second generation Raspberry Pi, running the Raspbian 8.0 operating system. The choice of Raspberry Pi for our implementation is dictated by its widespread acceptance as a microcomputer for constrained environments. It is worth noting that the Raspberry Pi Model B has a large amount of memory of 1GB of RAM and a 32 bit processor operating at speed of 700MHz. The Gateway provides two interfaces: a IEEE 802.11 short-range radio interface and an Ethernet interface. The short-range interface provides connectivity to the nodes in the IPv6 network while the Ethernet interface connects the Raspberry Pi to the IPv4 network.

Our implementation utilizes two different types of CoAP endpoints:

- *Intel Edison* boards with the *Ubilinux*¹ operating system running IPv6 over IEEE 802.11 with CoAP at the application layer. These devices are part of the IPv6 constrained network. The purpose of testing with these devices is to resemble as closely as possible the limited capabilities of the constrained devices.
- The IPv4 CoAP clients are running on an Ubuntu-14.04 desktop with Intel®Core™i7-950M processor at 3.07 GHz. They connect with the Gateway through the Ethernet interface. The IPv4 endpoints do not necessarily need to be constrained. Hence, we use a laptop with a wide spectrum of capabilities to simplify our implementation.

Some of the logical entities described in Figure 1 are software entities. These components are implemented as software running in the CoAP gateway or in the CoAP devices.

- The *Port Control Protocol (PCP)* client and server components were based on Alfred Heggestad's open source implementation² of the library. Since the library was incomplete, we implemented the missing parts for our prototype.
- *Libcoap*³ is an open source, multi-platform library implemented in C language. The library is designed to support all the features standardized in the IETF CoAP specification[21] and includes a basic CoAP server and client application. Unfortunately, it does not include a proper implementation of the resource directory [22]. We implemented a new resource directory based on the Libcoap library. We additionally integrated the source code of the PCP client into the resource directory implementation and modified it accordingly to support the NAT logic. The basic client and server CoAP applications were also modified to make the IPv6 devices capable of registering resources and listening at requests automatically.
- *NAT44 and NAT64* are two separate components. Whereas NAT44 is based on the *iptables* provided by the Linux kernel in the prototype, Tayga⁴ was the option to run the NAT64 component. Tayga is a user space stateless NAT64 implementation for Linux. Being a stateless NAT, Tayga requires that a unique IPv4 address is assigned to every IPv6 host that needs NAT64 service. In our prototype, such an assignment is done dynamically by Tayga from a pool of private IPv4 addresses. In addition to that, we modified the limit of the maximum lifetime of a mapping rule in the system to twenty-four hours to be aligned with the maximum lifetime of a resource in the resource directory.
- *BIND (Berkeley Internet Naming Daemon)* is the software used as the DNS64 server for the prototype.

5.1 Sample Scenario of the NAT Implementation

This section describes a sample scenario performed in our prototype in order to provide more technical details of our NAT implementation,

Our implementation is shown in Figure 4 and consist of a CoAP Gateway with two interfaces connected to an IPv4 network and an IPv6 network, respectively. The IPv6 network is a wireless interface. In this particular case, the Raspberry Pi provides a private IPv6 address range (shown as *fdde:4c60:038b:16b4::1/64* in Figure 4) through its Wi-Fi access point that allows 2^{64} hosts. Moreover, we configure a static IP address in the Raspberry Pi for the Wi-Fi interface. We set up the *fdde:4c60:038b:16b5::1* address for our implementation. In addition to the Wi-Fi interface, the Raspberry Pi is connected to an Ethernet interface. It can be noted from the figure that the Ethernet interface is a private IPv4 network. However, that interface connects indirectly to the Internet using NAT (not showed in the figure). This interface assigns to the Raspberry Pi via DHCP an IPv4 address from the available IP addresses in the *10.42.0.0/24* address pool. Although the IP address is

¹ <http://www.emutex.com> ² <https://github.com/alfredh/> ³ <https://libcoap.net/> ⁴ <http://www.litech.org/tayga/>

obtained dynamically and might change over time, Figure 4 assumes for simplicity that the IPv4 address of the Raspberry Pi is 10.42.0.77.

The prototype uses two physical Intel Edison boards connected through their integrated Dual Band Wi-Fi with the CoAP Gateway. However, Figure 4 only shows one Intel Edison board for the sake of clarity. The IPv6 address assigned to that board is *fdde:4c60:038b:16b4::2*. The boards do not come with any integrated sensor. Hence, we set up a temperature sensor in each board that allows us to monitor the temperature changes.

The implementation also uses a DNS64 server in the Wi-Fi interface to resolve AAAA queries from the IPv6-only host into IPv4-only queries and maintain the mapping between them. It also provides a *Fully Qualified Domain Name* (FQDN) for the CoAP Gateway to facilitate the communication between the IPv6-only hosts with the CoAP Gateway. The chosen FQDN for the CoAP Gateway is *rd.private.com* and every Intel Edison board stores that information in a specific configuration file. On the other hand, the nodes in the IPv4 network primarily use IP addresses rather than FQDNs since there is no address resolution mechanism deployed in that interface.

Figure 4 also illustrates in more detail the processes of registration, resource retrieval and information retrieval.

In the registration process, the IPv6 node (the Intel Edison in our case) registers its temperature resource using the registration operation defined in the resource directory[22]. The following operation shows an example of the registration:

```
POST coap://rd.private.com/rd?ep=node
Content-Format: 40
Payload: </sensors/temp>;rt="temperature-c";if="sensor"
```

The operation shows how the IPv6 endpoint with name *node* registers a new resource type (rt) *temperature*. The operation does not include a *lifetime* variable in the request and assumes the resource directory is accessible on its default port 5683. If no lifetime is included, a default value of one day will be assumed by the resource directory.

Once the registration request is sent, the NAT64 (Tayga) maps the IPv6 address into an IPv4 address. Since Tayga is a stateless NAT, it requires that a unique IPv4 address is assigned to every IPv6 host. In our prototype, the assignment is done dynamically by Tayga from a pool of IPv4 addresses designated for this purpose by us. The pool of addresses assigned in our prototype for this purpose is *192.168.0.0/16* that allow to have 65534 dynamic mappings. Tayga keeps the assignments usable for up to two hours since the last packet seen, however, we modified that behaviour to keep the assignments usable for up to one day to align the time-outs with the time-outs in the resource directory. Conveniently, Tayga retains all the assignments for up to two weeks as long as the address pool does not become empty.

Going back to our example, the NAT64 dynamically maps the IPv6 endpoint's address *fdde:4c60:038b:16b4::2* with the IPv4 address *192.168.0.2* from the address pool. At this point, the resource directory receives the request. The resource directory then might create a new resource and the corresponding NAT44 mapping, or might update an existing one. In addition, it stores internally the NAT44 mapping and the *lifetime* information of every endpoint to monitor and manage them properly. In the current example, the resource directory creates a new resource and assigns a new NAT44 mapping to the device since that specific endpoint does not exist in the resource directory. Specifically, the resource directory assigns the unused port *3000* to the *192.168.0.2* address. It creates the mapping in the NAT44 sending a PCP request to the PCP server through the PCP client. The PCP server is located in the CoAP Gateway and can be only accessed locally through the port *5351*. Although the entries in the resource directory will eventually time-out after their lifetime, the IPv6 endpoints can explicitly remove its entries from the resource directory using a delete operation.

Once the resource is registered, the resource directory can be used for discovering resources registered in it. Returning to our example in Figure 4, the IPv4 CoAP client can perform a resource lookup in order to obtain the information of all the registered endpoints.

```
GET coap://10.42.0.77:5683/rd-lookup/res?rt=temperature
```

The example above performs a lookup in the resource directory of all the registered resources that are identified by the *temperature* name. In our particular case, there is only one registered resource with that name in the resource directory. Thus, the resource directory returns the following answer:

```
GET 2.05 Content
Content-Format: 40
Payload: <coap://10.42.0.77:3000/temp>;rt="temperature"
```

The operation displays the NAT44 mapping of the specific resource identified by *temperature* name in the resource directory. Specifically, it displays the *10.42.0.77* address and port *3000* assigned to the registered IPv6 endpoint.

At this stage, the IPv4 CoAP client can use that information to directly communicate with the IPv6 endpoint inside the CoAP Gateway.

```
GET coap://10.42.0.77:3000/temp
```

The NAT44 automatically maps the port *3000* with the *192.168.0.2* address and immediately after, the NAT64 translates it to the *fdde:4c60:038b:16b4::2* address. Upon success, the IPv6 endpoint responds with the temperature. Thereafter, the NAT64 and NAT44 translate again the IPv6 address before to send the request back to the IPv4 CoAP client.

6 EVALUATION

Although our prototype successfully worked using two physical Intel Edison boards in the IPv6 network, we still deemed further tests to be necessary to evaluate the performance and scalability of the CoAP Gateway.

6.1 Experimental Setup

All the following experiments were done on a *Raspberry Pi 2 Model B* and two Ubuntu-16.04 laptops with Intel®Core™i7-950M processor at 3.07 GHz and 16GB of RAM. The CoAP gateway was installed in the Raspberry Pi as in Section 5. An IPv4 connection was established between one of the laptops and the CoAP gateway through a Local Area Network (LAN) while the other laptop was connected to the IPv6 Wi-Fi access point of the CoAP gateway. The CoAP gateway obtained global connectivity through the LAN.

We used a benchmark tool called CoAPBench⁵ in each laptop. CoAPBench uses Californium⁶ as CoAP implementation baseline to simulate several IPv6 and IPv4 endpoints in both ends. CoAPBench is, to the best of our knowledge, the only benchmark tool available for the CoAP protocol. However, the tool was too limited for our testing purposes and we extended it with new functionality. After our modifications, CoAPBench is able to send CoAP messages from different IPv6 addresses, it allows specifying a payload in the request messages and supports the *Resource Directory's* operations. These operations are described in Figure 2 as *resource registration*, *resource discovery* and *information retrieval* operations.

The *resource registration* operation is initialized by the IPv6 endpoint and ends once the resource directory creates the mapping in the NATs and returns a response code to the IPv6 endpoint. In order to use the resource directory to discover resources registered in it, a lookup interface is provided. The *resource retrieval* operation requests the resources of the resource directory using

⁵ <https://github.com/eclipse/californium.tools> ⁶ <https://eclipse.org/californium/>

a: Latency in the IPv4 network

b: Latency in the IPv6 network

Fig. 5. Latency

the *./well-known/core* operation. The *information retrieval* operation is similar to the last operation described in Figure 2. It retrieves the information of a sensor and the messages pass through both NATs.

CoAPBench was configured in such a way that every virtual client in the laptop connected to the IPv6 network had a different IPv6 address. Every IPv6 virtual client registered a new sensor into the CoAP gateway accordingly. We increased the number of IPv6 virtual clients in CoAPBench from 1 to 12,000 to simulate different network sizes. After registration into the CoAP gateway, every IPv6 virtual client stressed the CoAP gateway by continuously sending registration update requests during 60 seconds. To avoid bias, we repeated all the experiments five times.

We carried out similar experiments in the IPv4 network using the other laptop. In this case, however, the virtual clients of that CoAPBench used the same IPv4 address although different ports. In this test, every IPv4 virtual client discovered from the CoAP gateway the resource information registered previously by the IPv6 endpoints.

The experiments in our prototype were limited to the *registration*, *update* and the *resource discovery* operations, since the *information retrieval* operation only passed through the NATs which have not been modified in this prototype. For this reason, we limited the evaluation of our prototype to two scenarios: in the first scenario, we conducted experiments to simulated different network size where the virtual clients stress the CoAP gateway registering, updating and querying for the resources. The second scenario simulates a diverse number of concurrent virtual clients registering and discovering resources in the CoAP gateway. Both scenarios ran CoAPBench increasing the number of virtual clients from a range of 1 to 12,000 and both scenarios use a single CoAP gateway.

The goal was to gain an understanding of the scalability, as well as the performance and CPU/memory/energy footprint of the CoAP gateway. Gaining such an understanding is important, since all the information in the constrained IPv6 network go through the CoAP gateway and it is of utmost importance to quantify the capabilities of it.

6.2 Scalability of the CoAP Gateway

Our first experiment consist of evaluating the latency of the *registration* and the *resource retrieval* operations. We plotted the results into cumulative distribution functions (CDF) to have a better understanding of the distribution of latencies. Figure 5 shows the latencies of the virtual clients in the IPv4 network performing the *resource retrieval* operation (5a) and the latencies of the virtual clients in the IPv6 network performing the *registration* and *update* operations (5b).

a: Throughput of the CoAP Gateway

b: Timeouts

Fig. 6. Throughput: Diverse Number of Concurrent Virtual Client

It can be clearly seen that requesting resource information through the *resource retrieval* operation (5a) performs much better in terms of latency than registering and updating resources (5b) in the CoAP gateway. While the latency is relatively small for up to 100 concurrent clients, it drastically increase for higher concurrencies to the point that the average latency for 10,000 concurrent clients is 400 milliseconds. The reason behind is the complexity of the registration and update operations, both operations have to update the database in the *resource directory* and add the new NAT rules into the NAT44 using the PCP protocol. The PCP client queues all the PCP requests and sends them to the PCP server gradually to avoid overloading the PCP server and to prevent it from discarding any requests. Besides, the registration operation implies to modify the NAT64 as well. However, it is important to know that this experiment stressed the CoAP gateway performing a continuously registration update operations during 60 seconds once the virtual clients have been registered into the system. In a real case scenario, the registration and update operations are performed a limited number of times during the lifetime of an endpoint. Hence, we believe latencies of 100 milliseconds or higher would not affect the overall performance of the system.

In contrast, the retrieve operation in the IPv4 network produces an overall low latency even for a large number of concurrent virtual clients. This operation queries information from the *resource directory's* database.

Consequently, our system achieves good response times for the retrieve operation even for high concurrency factors and acceptable levels for the registration and update operations.

6.3 Performance of the CoAP Gateway

The second experiment evaluates the throughput of the CoAP gateway. The results of our evaluation are listed in Figure 6a, where the y-axis represents the transactions per second of the throughput and the x-axis represents the number of virtual clients. Figure 6a also analyses the impact in the throughput of simulated network latencies in both IPv4 and IPv6 networks. In addition to the scenario without artificial latency, we added latencies of 60ms and 150ms which are typical latencies for cloud services. We used a software called *tc* to generate the latencies.

The CoAP gateway achieved its peak performance with the resource retrieval operation (as well called service discovery) in the IPv4 network with 7164.44 request per second with 5,000 concurrent clients.

On the other hand, the registration and update operations in the IPv6 network achieved its peak performance with 1310.90 request per second with 5 concurrent clients. At this point, the performance dropped significantly to slightly less than 1,000 request per second with 500 concurrent

Fig. 7. Memory, CPU and Energy Footprint

clients. Afterwards, the performance declined steeply until 16.77 request per second for 12K concurrent clients.

Figure 6b shows the number of timeout requests. The graphs shows a consistent increase of the timeouts in relation to the number of virtual clients. However, it remains to zero for up to 50 concurrent clients.

The reason for the lower throughput for the registration or update operations is similar to the reason explained in the previous subsection. Both operations add or modify the NAT rules in the NAT44 using the PCP protocol. The PCP client implemented in the prototype do not implement concurrency, queuing the PCP request and sending them sequentially. The PCP operations incurs into big delays, which are necessary in order to keep the data in the NAT44 in a valid state.

6.4 Memory, CPU and Energy Footprint in the CoAP Gateway

We evaluated the memory consumption, CPU and energy utilization in the *Raspberry Pi 2 model B*, which has 1GB of RAM and 32 bit processor operating at speed of 700MHz. Even though the CPU utilization depends on the hardware, this evaluation aimed to test the resource utilization of a typical CoAP gateway device. In this test, 10,000 virtual clients registered a new resource into the CoAP Gateway. After registration, all the virtual clients stressed the CoAP gateway sending continuous *registration update* operations during 60 seconds. The test is similar to the *resource registration* test performed in section 6.2.

We used a software called *dstat* to generate the system resource statistics and a USB voltage meter to obtain the energy consumption values. Figure 7 shows the CPU, memory and energy consumption of the CoAP gateway during the test. The graphs in Figure 7 represent a 3-minutes period of time where the test started one minute after *dstat* and the USB voltage tester started collecting data. The test was executed during 60 seconds. Figure 7a shows that the CPU utilization was almost 100% during all the execution of the test while the maximum memory consumption (7b) was only 12MB (263MB-251MB, where 263MB was the maximum memory usage during the test and 251MB is the total used memory by the system before the test started). The energy consumption increased to 3.11W. After the test, the memory consumption increased to 5MB. Consequently, the different entities in the CoAP gateway utilized 5MB to store the information of 10,000 virtual clients with their respective sensor information and rules.

In conclusion, the *Raspberry Pi 2 model B* can successfully store the registration information of 10,000 virtual clients in 5MB of memory. On the other hand, the CPU consumption was very high during the test implying that the *Raspberry Pi* cannot achieved very high loads of registrations in

short period of times. However, even though the registration and update operations are expensive operations in terms of CPU, they are operations executed very seldom by the endpoints which reduce the chances of overloading the CoAP gateway.

7 CONCLUSIONS AND FUTURE WORK

This paper introduces a new transaction mechanism that enables CoAP endpoints outside a NAT to initiate communication with endpoints behind a NAT. A common problem with dynamic NATs is the inability for nodes outside the NAT to initiate the communication. As a result, the current NAT behaviour and the traditional NAT scenario –where the nodes behind a NAT originates the communication– prevents the proliferation of some IoT scenarios. Although several techniques have been developed during the years to overcome this limitation(e.g. ICE[18] and UDP hole punching), the majority of the existing solutions were developed for conventional Internet scenarios. IoT scenarios differ from conventional scenarios in terms of processing, memory and power capabilities. Moreover, due to the need for a large address space for a large number of devices, a considerable part of the IoT technologies are designed to use IPv6[8] in the future. However, despite the maturity of IPv6, the deployments of IPv6 have been relatively slow on the Internet. IoT networks are dependent entirely on address translation techniques since a considerable number of IoT technologies natively supports IPv6. For all these reasons, it is essential for the development of new techniques to overcome the challenges taking into consideration the stringent constraints of the IoT scenarios.

Therefore, the aim of this work is to provide an efficient and transparent solution for IoT scenarios that enables CoAP devices behind a NAT to receive inbound connections from other CoAP devices. As opposed to the existing NAT solutions, our solution enables the communication to be originated from outside the NAT and the NAT address translation is totally transparent to the CoAP devices optimizing the utilization of the resources in the constrained devices. To achieve this, our prototype makes use of an IoT entity called the *CoAP resource directory*[22] and the *port control protocol*[27] and modify them accordingly to create a new solution that is more suitable for IoT scenarios using CoAP.

Results of the evaluation study document the feasibility of the solution. Hence, the current prototype implementation demonstrates seamless integration to IoT networks in the prevalent IPv4 world. The throughput of the system was not affected by our solution and the constrained components deployed to test the solution performed extremely well. Despite that, a large number of simultaneous requests slowed down notably the response of the CoAP gateway. Moreover, it turns out that some ports assigned to the IPv6 nodes were preventing other applications in the CoAP gateway from using them. Our solution should be extended to prevent this from occurring by defining a set of ports that cannot be assigned by the resource directory.

Other areas of future work include enhancing the solution to suite distributed scenarios, as well as study the quantitative performance between the proposed approach and existing solutions.

Based on our work, we have identified a number of issues with the resource directory specification [22] and contributed our improvements to the related working group in the IETF.⁷

ACKNOWLEDGMENTS

The author would like to thank Kristian Slavov, Nicklas Beijar, Mario Di Francesco, Jouni Mäenpää, Miika Komu, Tero Kauppinen, and Jesus Llorente for all their feedback and help in this project.

REFERENCES

- [1] C. Aoun and E. Davies. 2007. Reasons to Move the Network Address Translator - Protocol Translator (NAT-PT) to Historic Status. RFC 4966 (Informational). (July 2007). <http://www.ietf.org/rfc/rfc4966.txt>

⁷ <https://www.ietf.org>

- [2] F. Audet and C. Jennings. 2007. Network Address Translation (NAT) Behavioral Requirements for Unicast UDP. RFC 4787 (Best Current Practice). (Jan. 2007). <http://www.ietf.org/rfc/rfc4787.txt> Updated by RFC 6888.
- [3] M. Bagnulo, P. Matthews, and I. van Beijnum. 2011. Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers. RFC 6146 (Proposed Standard). (April 2011). <http://www.ietf.org/rfc/rfc6146.txt>
- [4] M. Bagnulo, A. Sullivan, P. Matthews, and I. van Beijnum. 2011. DNS64: DNS Extensions for Network Address Translation from IPv6 Clients to IPv4 Servers. RFC 6147 (Proposed Standard). (April 2011). <http://www.ietf.org/rfc/rfc6147.txt>
- [5] C. Bormann, M. Ersue, and A. Keranen. 2014. Terminology for Constrained-Node Networks. RFC 7228 (Informational). (May 2014). <http://www.ietf.org/rfc/rfc7228.txt>
- [6] B. Carpenter. 2011. Advisory Guidelines for 6to4 Deployment. RFC 6343 (Informational). (Aug. 2011). <http://www.ietf.org/rfc/rfc6343.txt>
- [7] S. Cheshire and M. Krochmal. 2013. NAT Port Mapping Protocol (NAT-PMP). RFC 6886 (Informational). (April 2013). <http://www.ietf.org/rfc/rfc6886.txt>
- [8] S. Deering and R. Hinden. 1998. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460 (Draft Standard). (Dec. 1998). <http://www.ietf.org/rfc/rfc2460.txt> Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112.
- [9] R. Despres. 2010. IPv6 Rapid Deployment on IPv4 Infrastructures (6rd). RFC 5569 (Informational). (Jan. 2010). <http://www.ietf.org/rfc/rfc5569.txt>
- [10] Ericsson. 2017. *Ericsson Mobility Report: On the pulse of the networked society*. Technical Report. Ericsson. <https://www.ericsson.com/mobility-report>
- [11] Roy Thomas Fielding. 2000. *Architectural Styles and the Design of Network-based Software Architectures*. Ph.D. Dissertation. University of California, Irvine. http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
- [12] IEEE. 2017. IEEE 802.15 standard. (2017). <http://www.ieee802.org/15>
- [13] IGD. 2017. Internet Gateway Device (IGD). (2017). <https://openconnectivity.org/upnp/specifications>
- [14] N. Kushalnagar, G. Montenegro, and C. Schumacher. 2007. IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals. RFC 4919 (Informational). (Aug. 2007). <http://www.ietf.org/rfc/rfc4919.txt>
- [15] LWM2M. 2017. OMA Lightweight M2M, Technical Specification. (2017). <http://openmobilealliance.org/about-oma/work-program/m2m-enablers/> [Open Mobile Alliance].
- [16] R. Mahy, P. Matthews, and J. Rosenberg. 2010. Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). RFC 5766 (Proposed Standard). (April 2010). <http://www.ietf.org/rfc/rfc5766.txt>
- [17] MQTT. 2017. MQTT Specification. (2017). <http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.pdf> [OASIS Message Queuing Telemetry Transport (MQTT)].
- [18] J. Rosenberg. 2010. Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols. RFC 5245 (Proposed Standard). (April 2010). <http://www.ietf.org/rfc/rfc5245.txt> Updated by RFC 6336.
- [19] J. Rosenberg, R. Mahy, P. Matthews, and D. Wing. 2008. Session Traversal Utilities for NAT (STUN). RFC 5389 (Proposed Standard). (Oct. 2008). <http://www.ietf.org/rfc/rfc5389.txt> Updated by RFC 7350.
- [20] J. Rosenberg, J. Weinberger, C. Huitema, and R. Mahy. 2003. STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs). RFC 3489 (Proposed Standard). (March 2003). <http://www.ietf.org/rfc/rfc3489.txt> Obsolete by RFC 5389.
- [21] Z. Shelby, K. Hartke, and C. Bormann. 2014. The Constrained Application Protocol (CoAP). RFC 7252 (Proposed Standard). (June 2014). <http://www.ietf.org/rfc/rfc7252.txt>
- [22] Zach Shelby, Michael Koster, Carsten Bormann, Peter Van der Stok, and Christian Amsuss. 2018. *CoRE Resource Directory*. Internet-Draft draft-ietf-core-resource-directory. Internet Engineering Task Force. <https://datatracker.ietf.org/doc/html/draft-ietf-core-resource-directory> Work in Progress.
- [23] P. Srisuresh and M. Holdrege. 1999. IP Network Address Translator (NAT) Terminology and Considerations. RFC 2663 (Informational). (Aug. 1999). <http://www.ietf.org/rfc/rfc2663.txt>
- [24] Thread. 2017. Thread Network Protocol. (2017). <https://www.threadgroup.org/>
- [25] O. Troan and B. Carpenter. 2015. Deprecating the Anycast Prefix for 6to4 Relay Routers. RFC 7526 (Best Current Practice). (May 2015). <http://www.ietf.org/rfc/rfc7526.txt>
- [26] G. Tsirtsis and P. Srisuresh. 2000. Network Address Translation - Protocol Translation (NAT-PT). RFC 2766 (Historic). (Feb. 2000). <http://www.ietf.org/rfc/rfc2766.txt> Obsolete by RFC 4966, updated by RFC 3152.
- [27] D. Wing, S. Cheshire, M. Boucadair, R. Penno, and P. Selkirk. 2013. Port Control Protocol (PCP). RFC 6887 (Proposed Standard). (April 2013). <http://www.ietf.org/rfc/rfc6887.txt> Updated by RFC 7488.
- [28] Zigbee. 2017. Zigbee Alliance. (2017). <http://www.zigbee.org/zigbee-for-developers/zigbee>

Received August 2017; revised November 2017; accepted March 2018