2019 International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)

Enabling CoAP-Based Communication Across Network Boundaries: Challenges and Solutions

Oscar Novo
Department of Computer Science, Aalto University, Finland
NomadicLab, Ericsson Research, Finland
oscar.novo@ericsson.com

***Abstract*—The depletion of the pool of unallocated IPv4 addresses has introduced new challenges in the deployment of the Internet of Things across IPv6 and IPv4 networks. In previous work we developed a CoAP-based transition architecture specifically designed to overcome some of these challenges. In this respect, this article extends that work identifying the most common IoT scenarios and their requirements to successfully achieve seamless cross-domain communication in IoT. Furthermore, an analysis of our architecture and the state-of-the-art network transition technologies is conducted on the basis of those requirements. Among other aspects, the analysis provides new insights of the shortcomings of our architecture. Based on these shortcomings, the article proposes new extensions of the architecture in order to achieve seamless communication across the most common IoT scenarios.**

***Keywords*-Internet of Things; Network Transition Technologies; Ubiquitous Deployment;**

I. Introduction

The growth of Internet of Things (IoT) across the Internet, together with the strong requirements for scalability, ubiquitous communication and reliability will require new approaches to integrate IoT in the current architecture of the Internet. One of these approaches is related to defining solutions able to provide cross-domain communication between the Internet and the IoT networks.

Presently, the exhaustion of the IPv4 address space has led to the widespread adoption of IPv6 in the IoT. However, that adoption has brought up new challenges to the IoT scene, one of them being seamless communication of the IoT devices across IPv6 and IPv4 networks. If we consider that one of the main requirements of the IoT is its ubiquitous communication through the whole Internet, finding a solution for this problem is crucial for the success of the IoT.

Although some might argue that the interconnection of different IP networks has been tackled during the last two decades, in reality the constraints and limitations of many IoT scenarios prevent those solutions to be fully adopted. As a result, IoT development is restricted by current solutions, limiting the opportunities that might arise from successful cross-domain communication.

This article begins with a description of the key requirements to successfully achieve network transition for

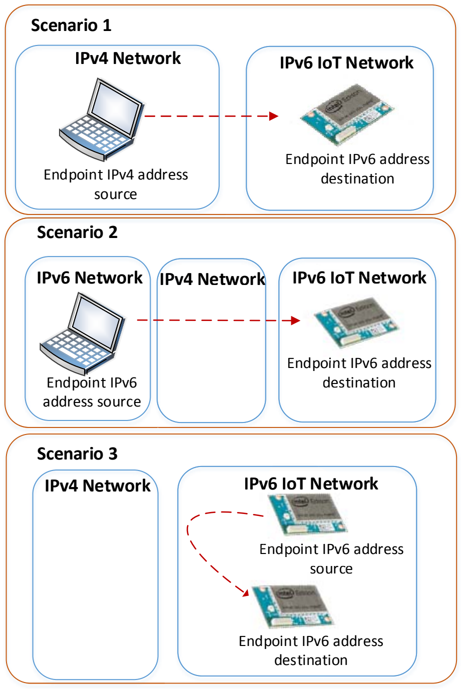

Scenario 1
IPv4 Network
Endpoint IPv4 address source
IPv6 IoT Network
Endpoint IPv6 address destination
Scenario 2
IPv6 Network
Endpoint IPv6 address source
IPv4 Network
IPv6 IoT Network
Endpoint IPv6 address destination
Scenario 3
IPv4 Network
IPv6 IoT Network
Endpoint IPv6 address source
Endpoint IPv6 address destination

Figure 1: Cross-Domain Communication Scenarios in IoT

Constrained Application Protocol (CoAP) [1] and CoAP-based communications in IoT. In addition, it identifies the most common IoT communication scenarios that network transition solutions should support to ensure seamless communication between IPv4 and IPv6. Then, the existing network transition technologies are concisely introduced and analysis with regard to those requirements.

In previous work we developed a novel transition technology specifically designed for CoAP-based communication [2]. In connection therewith, the second part of the article briefly introduces the architecture and compares it with the requirements previously presented. The analysis provides new insights of the shortcomings of the architecture. As a result of this evaluation, the article proposed new extensions of the architecture in order to achieve seamless communication across the most common IoT scenarios. Finally, a performance analysis is presented to show the effectiveness of the new modifications.

Our contribution This article features the following contributions:

- describing the most common cross-domain communication scenarios in IoT and providing the requirements to enable network transition in IoT;
- providing a succinct background of the current network transition technologies and analyzing their applicability with respect to IoT;
- identifying the shortcomings of our architecture and defining new solutions so that the architecture can handle multiple cross-domain communication scenarios in IoT;

II. Cross-Domain Communication in IoT

Before delving into the analysis of the network transition technologies, this section first defines the main scenarios that any network transition technology should support in IoT. In addition, this section describes the requirements that network transition technology should fulfill in order to operate properly in CoAP-based communications in IoT.

A. Description of the Most Common IoT Scenarios

As noted above, one of the consequences of the widespread adoption of IPv6 in IoT has been the depletion of the IPv4 pool of address. The adoption of IPv6, together with the requirement in IoT to achieve ubiquitous communication through the whole Internet, have portrayed a set of new communication scenarios in IoT. Figure 1 lists more succinctly the scenarios that any IoT communication should accomplish in order to achieve ubiquitous communication on the Internet.

Due to the lack of support of IPv6 on the Internet, it will be often necessary for the IoT devices to interact with IPv4 nodes or across IPv4-only networks. In addition, based on the design principles of IoT, in most cases the interaction will initiate from the IPv4 networks. This is particularly true for all the IoT devices that act as sensors and reside on IPv6 networks. In such cases, devices from outside the IPv6 network will request the information of the IoT sensors. This scenario is specified as *scenario 1* in Figure 1.

In some situations, the communication might need to traverse different network domains. For instance, two IPv6 networks might be separated by an IPv4 network, hindering the communication between the nodes belonging to the different IPv6 networks. Since the deployment of IPv6 is not yet widespread through the whole Internet, this situation would be a plausible scenario for IoT. This scenario is defined as *scenario 2* in Figure 1.

Another explicit situation in IoT is direct communication between the nodes belonging to the same IoT network. Different nodes in the same IoT network have to be able to discover the IoT devices and communicate with them on that network. Even though this requirement seems easy to accomplish by the majority of the networks, the IoT networks require a set of tools to achieve it, and occasionally those tools do not work properly through different domain networks. This scenario is defined as *scenario 3* in Figure 1. The three scenarios described in Figure 1 address the majority of IoT interactions on the Internet. For this reason, it is of utmost significance that the transition technologies used in IoT support all of them.

B. Requirements for Enabling Network Transition in IoT

Many CoAP-based IoT devices are characterized by low memory, battery and processing power, limiting the number of tasks that they can perform. Those limitations may negatively impact the ability of different network transition technologies to perform properly in some specific IoT scenarios. In certain scenarios, those limitations may even prevent the use of the network transition technologies completely.

In order to enable network transition in IoT, the network transition technologies should have the following characteristics to fulfill the basic requirements of the IoT devices:

- *Lightweight:* since IoT devices are constrained in nature, the transition technology solutions should be designed to interact as little as possible with the IoT devices.
- *CoAP compliant:* the most constrained IoT devices can only communicate through a constrained protocol stack. Therefore, the different solutions should be compliant with the IoT standards in general and CoAP in particular or should be designed to integrate them.
- *Transparent:* transparency or the actual invisibility of the communication process towards the IoT devices is especially desirable in the network transition technologies due to the limited capabilities of the IoT infrastructures. In addition, transparency increases the interoperability of the transition technologies in the IoT scenarios.
- *Cross-platform communication:* the IoT scenarios should be able to communicate indistinctly with devices and services from various address families. In other words, the network transition technologies should not limit the communication between networks belonging to different address families.
- *Secure:* the IoT devices should be able to use end-to-end security. In addition, unsolicited inbound network traffic should not reach the IoT devices.
- *Scalable:* the different network transition technology solutions should be able to map a single resource among several IoT devices. For instance, multiple devices sharing a single public IPv4 address.
- *Prevent protocol overhead:* the different transition technologies should avoid adding protocol header overhead.

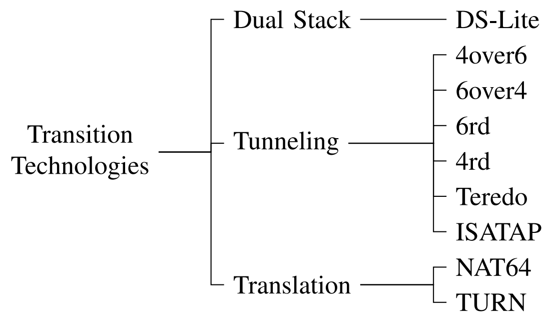


Figure 2: Network Transition Technologies

Hence, the system sending the information across the network has to fragment the packets.

- *Allow inbound connections:* the different transition technologies should allow the IoT devices to receive inbound communication from devices belonging to different networks or address families.

The characteristics listed above should be fulfilled by any network transition technology in order to achieve proper integration with the existing IoT scenarios.

III. Network Transition Technology Landscape

The current transition technologies can be placed in three different categories as depicted in Figure 2: tunneling, translation and dual-stack. The *tunneling* technology provides a method of transporting IP packets between networks that do not support the same Internet protocol, by encapsulating them. One of the disadvantages of tunneling is the inability to communicate between networks that support different Internet protocols. This is also one of the main differences between the *tunneling* and the *translation* technologies. The *translation* technology translates between different versions of Internet Protocols and, therefore, differs from the tunneling technology which enables direct communication between networks belonging to different address families. However, this technology has one major drawback: it breaks the end-to-end principle of Internet. On the other hand, the *dual-stack* technology implements both IPv4 and IPv6 protocols on the same node. The main challenge introduced by this technology is the overhead of implementing both stacks in every node.

Various transition technologies have been introduced during the last decades. Figure 2 lists the most common ones for each category. In addition, a brief description is provided below. The description is not designed to be very detailed since much information already exist in this area:

- *4over6:* IPv4-over-IPv6 [3] was designed to provide IPv4 Internet connectivity over an IPv6 access network using global IPv4 addresses.
- *6over4:* IPv6-over-IPv4 [4] provides connectivity between different IPv6 islands by encapsulating IPv6 packets inside IPv4 packets.
- *6rd:* IPv6 Rapid Deployment on IPv4 Infrastructures (6rd) [5] specifies an automatic tunneling mechanism tailored to advance deployment of IPv6. 6rd transfers IPv6 packets over the IPv4 network.
- *4rd:* IPv4 Residual Deployment via IPv6 (4rd) [6] is a technology for deployment of IPv6. In this mechanism, IPv4 packets are transparently tunneled across IPv6 networks.
- *Teredo:* In some specific cases, the 6over4 technology does not work due to its inability to cross Network Address Translation (NAT) boundaries. Teredo [7], on the other hand, is a NAT traversal technology for IPv6 traffic designed to cross NAT boundaries.
- *ISATAP:* The Intra-Site Automatic Tunnel Addressing Protocol (ISATAP) [8] is a mechanism intended to transmit IPv6 packets between dual-stack nodes on top of an IPv4 networks.
- *DS-Lite:* Due to the depletion of the IPV4 addresses, Internet Service Providers (ISP) need new addresses to supply their customers. Providing IPv6 addresses alone is not workable since most of the networks still enable IPv4 hosts and services. Dual-stack Lite [9] provides a solution where ISPs can migrate to an IPv6 access network while maintaining IPv4 communication through a dual-stack. The IPv4 packets are encapsulated over the IPv6 packets. ISPs uses their own IPv6 access networks to deliver the packet. The original IPv4 packet is recovered and routed to the public IPv4 Internet.
- *NAT64:* The Network Address Translation is a method of remapping an IPv4 address into an IPv6 address [10]. Networks Address Translations can be classified as stateless and stateful. Stateless translations achieve a one-to-one address mapping while stateful translations multiplex many IPv6 addresses into a single IPv4 address.
- *TURN:* Traversal Using Relays around NAT (TURN) [11] is a mechanism to rely messages between two devices behind a NAT through a third-party server. TURN aims at devices running multimedia applications.

The current transition classification is not inherently limited to one category. Many of the transition technologies described above can be placed in different categories. For example, DS-Lite employs dual-stack and a translation mechanism at the same time.

A. Applicability Analysis for IoT

As Table I shows, the current transition technologies lack the ability to provide lightweight deployments for the IoT scenarios. This characteristic is of utmost importance to constrained networks with limited resources. In addition, the majority of the technologies – except for the technologies that use translation gateways or third-party proxies – need

	Lightweight	CoAP Compliant	Transparent	Cross-Platform Communication	Secure	Scalable	Protocol Overhead	Inbound Connections	Scenarios		
	Network	Device	Device	Device	Device	Device	Network	Network	1	2	3
4over6	needs encapsulation	✓	High	Only IPv4 networks	unsolicited inbound traffic	1-to-1	40 bytes	✓	✗	✗	✗
6over4	needs encapsulation	✓	High	Only IPv6 networks	unsolicited inbound traffic	1-to-1	20 bytes	✓	✗	✓	✓
6rd	needs encapsulation	✓	High	Access IPv6 services	unsolicited inbound traffic	1-to-N	20 bytes	✓	✗	✓	✓
4rd	needs encapsulation	✓	High	Access IPv4 services	unsolicited inbound traffic	1-to-N	40 bytes	✓	✗	✗	✗
Teredo	translation gateways	✓	High	Access IPv6 services	broken end-to-end	1-to-N	0 bytes	✓	✗	✓	✓
ISATAP	tunneling	✓	High	Access IPv6 services	unsolicited inbound traffic	1-to-1	20 bytes	✓	✗	✓	✓
DS-Lite	dual stack	✓	High	✓	unsolicited inbound traffic	1-to-1	40 bytes	✓	✓	✗	✓
Stateful NAT64	translation gateways	✓	High	✓	broken end-to-end	1-to-N	0 bytes	✗	✗	✗	✓
Stateless NAT64	translation gateways	✓	High	✓	broken end-to-end	1-to-1	0 bytes	✗	✗	✗	✓
TURN	third-party servers	✗	Low	✓	broken end-to-end	1-to-1	0 bytes	✗	✗	✗	✓

Table I: Gap Analysis of the Existing Transition Technologies

Lightweight	CoAP Compliant	Transparent	Cross-Platform Communication	Secure	Scalable	Protocol Overhead	Inbound Connections	Scenarios		
Network	Device	Device	Device	Device	Device	Network	Network	1	2	3
third-party servers	✓	High	✓	✓	1-to-N	0 bytes	✓	✓	✗	✗

Table II: The COAP-Based Transition Architecture

to encapsulate the different messages adding protocol overheads into them. Thus, the networks have to transport more data to represent the same information, and they have to use extra resources to cipher and decipher the information at the edge. Besides that, the translation technologies break the end-to-end communication between endpoints, while the rest of the technologies accept unsolicited inbound traffic. Accepting unsolicited inbound traffic might look like a minor security issue but constrained devices with limited capabilities cannot deal easily with this traffic consuming their limited resources. On the other hand, the translation technologies do not allow inbound connections from devices outside the NAT, limiting the usability of those technologies in the IoT scenarios. In the majority of the IoT scenarios the constrained devices act as servers and the communication is originated from non-constrained devices, meaning that the communication originates outside the constrained network.

Aside from the security and protocol overhead, the majority of the technologies score high in transparency and CoAP compliancy. This is due to the fact that the majority of these technologies are protocol-independent, and any protocol can run on top of them, including the IoT protocols such CoAP [1]. An exception to this rule is TURN. In TURN, the devices need to communicate directly with the TURN servers, and the current specification [11] is not CoAP compliant. Further, some technologies have the ability to map several devices using a single resource, minimizing the number of resources used in the network and increasing the scalability of the network. For instance, while the *stateful NAT64* maps several devices using a single IP address (1-to-N in the table), the *stateless NAT64* makes use of a different IP mapping for every device (1-to-1 in the table). This feature is paramount in constrained networks, and the different transition technologies for IoT should provide this feature to prevent wasting the resources of the network.

In addition, Table I summarizes the feasibility of the different transition technologies to support the specific IoT scenarios. As can be noted in Table I, the current transition technologies do not support all the IoT scenarios, precluding the use of these technologies in IoT.

IV. The CoAP-Based Transition Architecture

In previous work [2], we have introduced a novel transition technology for IoT. The architecture is a new translation mechanism for CoAP [1] devices based on network address translation (NAT). Figure 3 illustrates the architecture defined in [2], its components and interactions. The new architecture acts as a CoAP gateway between an IPv4 network and an IPv6 constrained network. The role of the gateway is to act as a translation proxy between two networks supporting a different network-layer infrastructure. The gateway provides a NAT and a DNS64 mechanism. In addition, the gateway integrates an IoT entity called *Resource Directory* [12]. The *Resource Directory* was originally designed to discover the IoT resources of a network. However, the *Resource Directory* has not been designed with the intention of running among different versions of IP networks. The proposed architecture adapts the *Resource Directory* in a way that IPv4 devices can seamlessly discover the resources in an IPv6 network and communicate with those resources transparently. In addition, the *Resource Directory* includes a new logic that allows it to dynamically monitor, store, and manage the NAT rules in the gateway through a *Port Control Protocol* (PCP) [13] client. Thus, the *Resource Directory* keeps track of the resources and creates static rules in the NAT44 using a PCP Client.

After an IoT endpoint has registered its resources to the *Resource Directory* (blue interaction line in Figure 3), the *Resource Directory* instructs the NAT to assign a new port to those resources and stores that information internally (red interaction line in Figure 3). Once the resources are registered, any device can request the information of those resources through the *Resource Directory*'s REST interface. The *Resource Directory* returns the information of the resource and the IP and port provided by the NAT. Thereafter, the device can communicate directly with the IoT device without being aware of the existence of a NAT mechanism between them (green interaction line in Figure 3). At any time, the IoT devices can modify the information of their resources in the *Resource Directory*, and that will be reflected in the NAT. For instance, if an IoT device deletes its resources from the *Resource Directory*, those resources would be automatically deleted from the NAT, too.

The scenario depicted in Figure 3 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 endpoint and its communication with the IPv6 endpoint associated to that resource.

It should be noted that our implementation combines a NAT44 [14] and a NAT64 [10]. Our implementation uses a stateless NAT64 which performs one-to-one substitutions of IP addresses. In addition, we enhanced the solution adding a NAT44. Whereas NAT44 is based on the *iptables* provided by the Linux kernel in the prototype, Tayga[1] 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. As a result, the NAT44 only requires a single external IPv4 address to map all the internal communication reducing significantly the number of address translations used by the CoAP gateway.

[1] http://www.litech.org/tayga

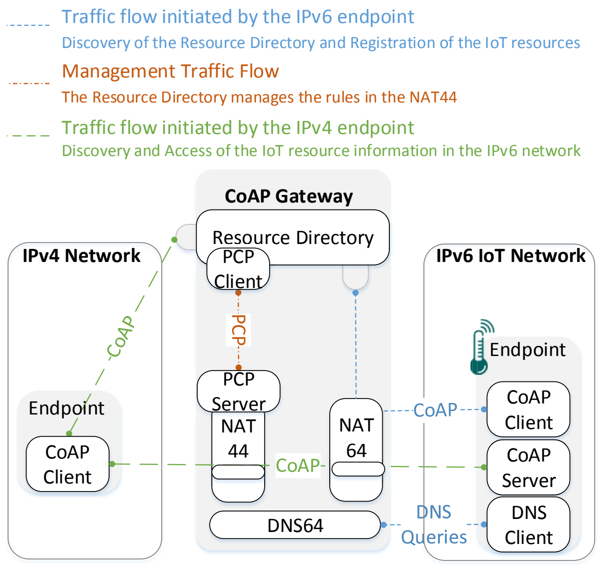

Traffic flow initiated by the IPv6 endpoint
Discovery of the Resource Directory and Registration of the IoT resources
Management Traffic Flow
The Resource Directory manages the rules in the NAT44
Traffic flow initiated by the IPv4 endpoint
Discovery and Access of the IoT resource information in the IPv6 network
CoAP Gateway
Resource Directory
PCP Client
IPv4 Network
IPv6 IoT Network
CoAP
PCP
Endpoint
PCP Server
NAT 44
NAT 64
CoAP
CoAP Client
CoAP Server
CoAP
CoAP Client
DNS64
DNS Queries
DNS Client
Endpoint

Figure 3: Logical Architecture of the NAT Implementation

The rest of the software components described in Figure 3 are implemented as software running in the CoAP gateway or in the CoAP devices. The software used as the DNS64 server is *BIND*[2] *(Berkeley Internet Naming Daemon)*. The *Port Control Protocol (PCP)* client and server components were based on Alfred Heggestad's open source implementation[3]. We implemented a new *Resource Directory* based on the *Libcoap*[4] library and additionally integrated the source code of the PCP client into the *Resource Directory* implementation and modified it to support the NAT logic. The CoAP clients and servers were also implemented using *Libcoap*.

A. Applicability Analysis for IoT

The main goal of our architecture was to overcome the shortcomings of the current transition technology identified in the previous section. As shown in Table II, the new architecture is able to solve many of the limitations of the current transition solutions. Overall, the new architecture has the following benefits in comparison with the current transition technologies:

- *Lightweight:* The architecture is inherently lightweight due to the integration of different IoT standards into the solution. Furthermore, the solution does not add any *protocol overhead* to the messages.
- *CoAP compliant:* The architecture adopts CoAP and the *Resource Directory*, two standards specifically designed

[2] https://www.isc.org/downloads/bind
[3] https://github.com/alfredh/
[4] https://libcoap.net

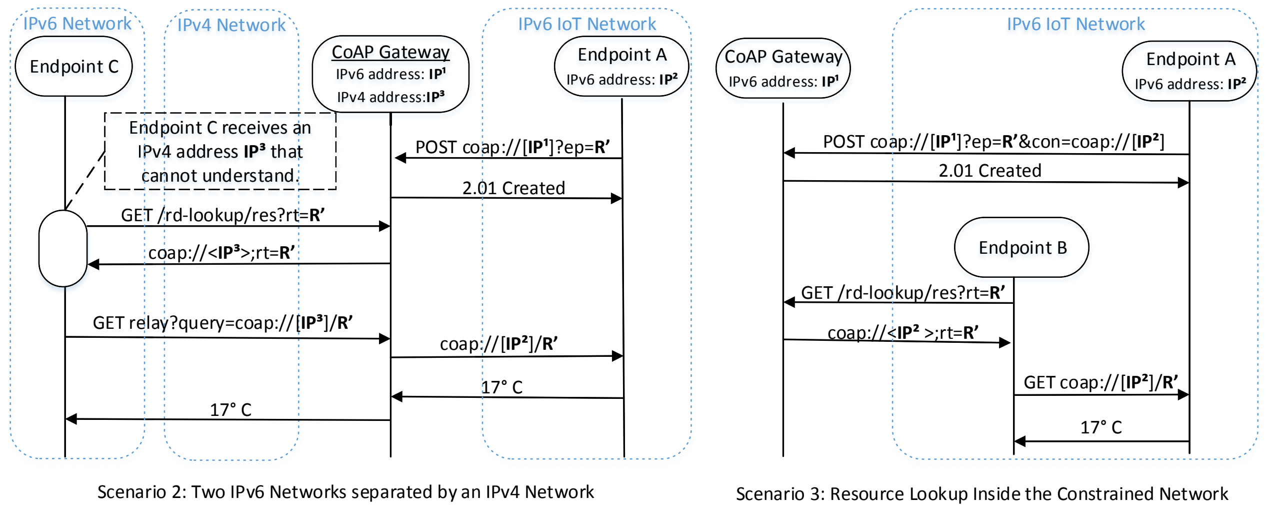


Figure 4: Cross-Domain Communication Extensions: Scenario 2 (left) and Scenario 3 (right)

for constrained scenarios. Currently, both standards are widely adopted by the industry.

- *Transparent:* The transition mechanism is totally transparent to the devices, easing the integration of very constrained devices into the solution. The devices just interact with each other following the standards as in any other scenario.
- *Cross-platform communication:* The architecture allows communication between devices belonging to different address families.
- *Scalable:* Additionally, the architecture is designed to reuse the resources in the NAT. Thus, different IoT devices can use the same NAT resource.
- *Allow Inbound Connections:* the devices behind the NAT can receive *inbound connections*. The architecture also accepts a certain degree of *security*, restricting unsolicited traffic to devices behind the NAT that do not wish to receive any inbound traffic.

Following all these benefits, the new architecture is able to solve many of the limitations of the current transition solutions. However, although the proposed architecture provides greater benefits than the current transition technologies, the architecture was mainly designed for the scenario where IPv4 devices initiated the communication with IPv6 constrained devices (*scenario 1* of Figure 1). The rest of the scenarios defined in Figure 1 (*scenario 2* and *scenario 3*) were not successfully achieved by it. Therefore, our architecture is still limited. The limited number of scenarios in which the architecture can be applied, prevents its broader adoption in other CoAP communication models.

Next section proposes new modifications of the architecture to enable CoAP-based communication in the rest of scenarios.

B. New Extensions of the Current Architecture

This section describes the new enhancements incorporate to the architecture to support *scenario 2* and *scenario 3* defined in Figure 1.

Scenario 2 – IPv6 Networks Separated by an IPv4 Network

One of the limitations of the former architecture was the inability of endpoints behind a NAT64 to initiate communication with the IoT resources in the constrained network (*scenario 2* in Figure 1). If the IP provided by the *Resource Directory* did not belong to the same network domain where the endpoint was located, the endpoint could not request the resource's information directly from the resource. In this particular case, the lookup information received by the *Resource Directory* was invalid if the endpoint was behind a NAT64.

The solution for this scenario is to proxy the information through the *Resource Directory* in order to help forward the resource information to the endpoint behind the NAT64. The left part of Figure 4 shows an example of this extension. First, *Endpoint A* who is a constrained device in the IPv6 IoT network, registers the resource *temperature* in the *Resource Directory* of the CoAP gateway. The resource *temperature* is represented as R' in Figure 4. Afterwards, *Endpoint C* determines that is behind a NAT. An endpoint can determine that is behind a NAT if it cannot interpret the lookup information obtained from the *Resource Directory*. The following operation shows an example of a lookup of a resource by *Endpoint C*:

```
Req: GET /rd-lookup/res?rt=temperature

Res: 2.05 Content
     <coap://[123.123.23.1]:5683/temp>;rt="temperature"
```

If the IP provided by the *Resource Directory* does not belong to the same network domain where the endpoint is located, the endpoint cannot request the resource's information directly from the resource. In the message above (shown also in Figure 4), the *Resource Directory* returns an IPv4 address when the *Endpoint C* belongs to an IPv6 domain network.

Once *Endpoint C* concludes that is behind a NAT, it can indicate willingness to get the information of a specific resource through the *Resource Directory*. Thereafter, the *Resource Directory* requests the information of the resource on behalf of the *Endpoint C*. The message below describes the new *relay* operation and its new *query* parameter that specifies the IP of the resource. That IP has been provided previously by the *Resource Directory* in the lookup operation, represented as IP^3 in Figure 4.

```
Req: GET /relay?query="coap://[123.123.23.1]:5683/temp"

Res: 2.05 Content
     17
```

Upon receiving the *relay* message from *Endpoint C*, the *Resource Directory* changes the IPv4 address of the request (IP^3 in Figure 4) to the IPv6 address of endpoint A in the constrained network (IP^2 in Figure 4). Once the *Resource Directory* obtains the information from *Endpoint A*, it forwards it to *Endpoint C*: 17°C.

Even though this solution aims to facilitate the nodes behind a NAT to query the information of the resources in the constrained network through the *Resource Directory*, particular attention should be paid to the cost of this operation in the *Resource Directory*. Therefore, this operation should be limited to avoid disrupting the performance of the *Resource Directory* and the CoAP gateway in general.

Scenario 3 – Resource Lookup Inside the Constrained Network

Another limitation of the former architecture was the inability of the constrained devices behind the CoAP gateway to query other resources in their own network (*scenario 3 in Figure 1*). In this particular case, if a constrained device performed an IP address lookup of a resource in the same network, the *Resource Directory* returned an IPv4 address instead of the required IPv6 address.

As we explained in the previous section, the CoAP gateway combines a NAT44 and a NAT64 to minimize the number of IPv4 address translations employed by the CoAP gateway. Therefore, all communication between the constrained network and the *Resource Directory* goes through the NAT64 causing the *Resource Directory* to receive the NATted IP address for all communication initiated by the constrained network. Consequently, the *Resource Directory* was not able to provide the original IPv6 address of the resources inside the constrained network. This limitation has been solved including the IPv6 address from the endpoints behind the CoAP gateway in the registration operation. Then the *Resource Directory* stores that information and returns it to all query operations performed inside the constrained network. This can be achieved using the *con* parameter. The *con* parameter is currently optional in the registration operation. Hence, the solution proposed for this scenario is to modify the *Resource Directory* to return a new error code *4.10 "Missing IP Address"* in case the *con* parameter is missing from the registration request. In this manner, once an endpoint receives a *4.10 "Missing IP Address"*, it will interpret it as if the request has to be resubmitted again including the *con* parameter. The following operation shows an example of a failed registration missing the *con* parameter:

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

Res: 4.10 Missing IP Address
```

After the error message, the endpoint can interpret that the *con* parameter is missing and send a new registration operation including the *con* parameter and its IPv6 address on it:

```
Req: POST coap://rd.private.com/rd?ep=node&con=coap://[
     ↪ ff35:30:2001:db8::1]
     Content-Format: 40
     Payload:
       </sensors/temp>;rt="temperature-c";if="sensor"

Res: 2.01 Created
     Location: /rd/4521
```

An example of this scenario is depicted in Figure 4. *Endpoint A* registers its resource in the *Resource Directory* and includes its own IPv6 address under the *con* parameter, represented as IP^2 in Figure 4. When later *Endpoint B* searches for the resource *R'*, the *Resource Directory* returns the IPv6 of *Endpoint A*. Thereupon, *Endpoint B* contacts *Endpoint A* using its IPv6 address.

C. Evaluation

The new features of the architecture have been evaluated in terms of performance and scalability.

Evaluation Setup

All the experiments were done using a Raspberry Pi and two laptops with Intel®Core^{TM}i7-950M processor at 3.07 GHz. The CoAP gateway was installed in the Raspberry Pi. One of the laptops and the CoAP gateway were connected through wired connections to an IPv4 Local Area Network (LAN). The other laptop was connected to the IPv6 Wi-Fi access point of the CoAP gateway. The laptop connected to the IPv6 Wi-Fi access point of the CoAP gateway acted as the IPv6 IoT network. The other laptop represented the IPv6 network of *scenario 2* in Figure 1.

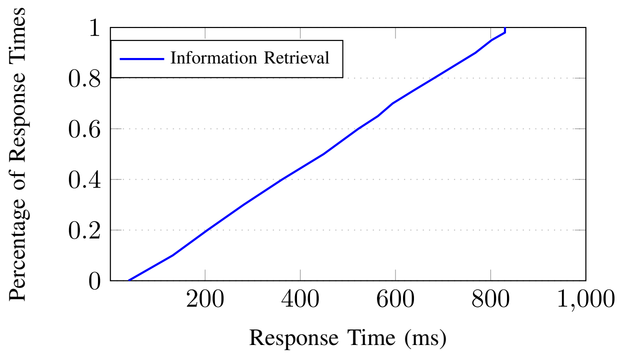
The new extensions of *scenario 2* and *scenario 3* have been evaluated in terms of the latency and throughput of the CoAP gateway. Primarily, we evaluated the new *relay* operation of the *Scenario 2*. This operation is initiated by the endpoint of the IPv6 network and ends once the *Resource Directory* returns the requested resource value to the endpoint. For the sake of simplicity, we will refer to this operation as *information retrieval*. Furthermore, we evaluated the new *con* parameter in *scenario 3*. The *con* parameter is part of the resource registration operation in *scenario 3*. This operation is initiated by an endpoint in the IPv6 IoT network and ends once the *Resource Directory* creates the mapping in the NAT and returns a response code to the endpoint.

The CoAP endpoints were simulated in each laptop through a benchmark tool called CoAPBench[5]. CoAPBench supports the Resource Directory's operations as well as all the CoAP message types.

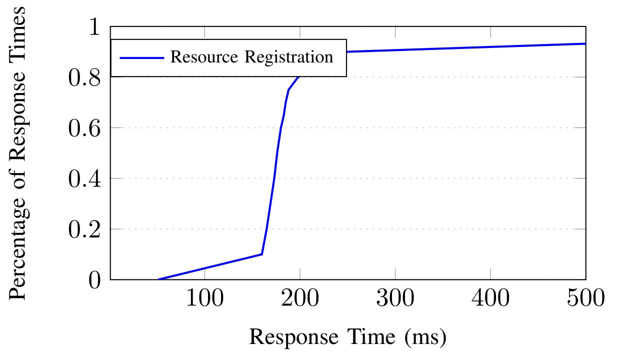
Results

The latency of the *information retrieval* operation was evaluated simulating 1,000 concurrent virtual clients sending *relay* operations to the *Resource Directory* and waiting for its answer. The results are plotted into a cumulative distribution function (CDF) in Figure 5a. It can be seen from the figure that the latency is relatively high, the *information retrieval* operation incurs into big delays. The *Resource Directory* has to modify every operation before forwarding it to the IoT endpoint behind the NAT and has to wait for the answer of each IoT endpoint before sending the answer back to its requester. In general, this operation is very costly for the CoAP gateway and should be used with due caution. As a matter of fact, we could not evaluate the throughput of this operation due to the large consumption of resources in the CoAP gateway.

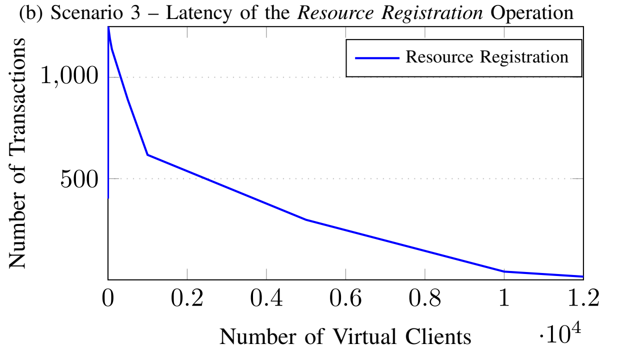
Figure 5b shows the latency of the *scenario 3*. In this scenario, we evaluated the *registration operation* simulating 1,000 concurrent virtual clients registering and updating resources in the CoAP gateway. All the registration operations contained the *con* parameter. Even though the *resource registration* operation performs much better in terms of latency than the *information retrieval* operation, it was still significantly high. This is because of the complexity of the registration and update operations. The operation has to update the database in the *Resource Directory* and add the new NAT rules into the NAT44 using the PCP protocol. Besides, the PCP client deployed in the prototype does not implement concurrency, queuing the PCP requests and sending them sequentially. Nonetheless, the registration and update operations are performed a limited number of times during the lifetime of an endpoint in a real case scenario. Hence, we believe those latencies would not affect the overall performance of the system. Figure 5c lists the results of the throughput. The throughput has been evaluated simulating various network sizes where a diverse number of virtual clients stress the CoAP gateway registering resources. The number of virtual clients increase from a range of 1 to 12,000. The CoAP gateway achieved its peak performance with 1,311 request per second with 5 concurrent clients. At this stage, the throughput slightly dropped to 1,138 request per second with 1,000 concurrent clients. Thereafter, the throughput declined to 16 request per second for 12,000 concurrent clients. The reason for the low throughput is similar to the reason explained before, the operation adds or modifies the NAT rules in the NAT44 using the PCP protocol. The PCP operations incur big delays which are



(a) Scenario 2 – Latency of the *Information Retrieval* Operation



(b) Scenario 3 – Latency of the *Resource Registration* Operation



(c) Scenario 3 – Throughput of the *Resource Registration* Operation

Figure 5: Performance and Scalability of the CoAP Gateway

[5] https://github.com/eclipse/californium.tools

necessary to keep the data in the NAT44 in a valid state.

V. Conclusion and Future Directions

In this article, we have listed the existing state-of-the-art transition technologies and evaluated them via a concise gap analysis that outlined their capacity to deploy CoAP-based IoT scenarios. To facilitate the discussion of our analysis, we first presented the key requirements to successfully achieve network transition for CoAP-based communication in IoT. In addition, we described the most common IoT communication scenarios where network transition solutions should be supported to ensure seamless communication across different address families.

In previous work [2], we developed a network transition architecture designed to facilitate the deployment of the IoT scenarios across different address families. In particular, the architecture defines an efficient IP network transition method for CoAP devices based on network address translation. The proposed architecture allows to seamlessly discover the IoT resources deployed behind a NAT in a constrained network and enables the communication among them.

Although the architecture was designed to overcome the current limitations of the existing network transition technologies in IoT, the analysis performed in this article uncovered some restraints of the architecture. We described them in detail and proposed new enhancements of the architecture to solve them. The new features have also been tested and evaluated.

Future research directions may include an integration of other lightweight application layer protocols in the current architecture. The current architecture is only integrated with CoAP at present, limiting its usability and accessibility as a result.

Acknowledgment

The author would like to thank Nicklas Beijar, Roberto Morabito, and Mario Di Francesco for their feedback.

References

[1] Z. Shelby, K. Hartke, and C. Bormann, "The Constrained Application Protocol (CoAP)," RFC 7252, RFC Editor, Fremont, CA, USA, pp. 1–112, Jun. 2014, updated by RFC 7959. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7252.txt

[2] O. Novo, "Making Constrained Things Reachable: A Secure IP-Agnostic NAT Traversal Approach for IoT," *ACM Trans. Internet Technol.*, vol. 19, no. 1, pp. 3:1–3:21, Oct. 2018. [Online]. Available: http://doi.acm.org/10.1145/3230640

[3] Y. Cui, J. Wu, P. Wu, O. Vautrin, and Y. Lee, "Public IPv4-over-IPv6 Access Network," RFC 7040, RFC Editor, Fremont, CA, USA, pp. 1–13, Nov. 2013. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7040.txt

[4] S. Steffann, I. van Beijnum, and R. van Rein, "A Comparison of IPv6-over-IPv4 Tunnel Mechanisms," RFC 7059, RFC Editor, Fremont, CA, USA, pp. 1–41, Nov. 2013. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7059.txt

[5] W. Townsley and O. Troan, "IPv6 Rapid Deployment on IPv4 Infrastructures (6rd) – Protocol Specification," RFC 5969, RFC Editor, Fremont, CA, USA, pp. 1–18, Aug. 2010. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5969.txt

[6] R. Despres, S. Jiang (Ed.), R. Penno, Y. Lee, G. Chen, and M. Chen, "IPv4 Residual Deployment via IPv6 - A Stateless Solution (4rd)," RFC 7600, RFC Editor, Fremont, CA, USA, pp. 1–45, Jul. 2015. [Online]. Available: https://www.rfc-editor.org/rfc/rfc7600.txt

[7] C. Huitema, "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)," RFC 4380, RFC Editor, Fremont, CA, USA, pp. 1–53, Feb. 2006, updated by RFCs 5991, 6081. [Online]. Available: https://www.rfc-editor.org/rfc/rfc4380.txt

[8] F. Templin, T. Gleeson, and D. Thaler, "Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)," RFC 5214, RFC Editor, Fremont, CA, USA, pp. 1–15, Mar. 2008. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5214.txt

[9] A. Durand, R. Droms, J. Woodyatt, and Y. Lee, "Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion," RFC 6333, RFC Editor, Fremont, CA, USA, pp. 1–32, Aug. 2011, updated by RFC 7335. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6333.txt

[10] M. Bagnulo, P. Matthews, and I. van Beijnum, "Stateful NAT64: Network Address and Protocol Translation from IPv6 Clients to IPv4 Servers," RFC 6146, RFC Editor, Fremont, CA, USA, pp. 1–45, Apr. 2011. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6146.txt

[11] R. Mahy, P. Matthews, and J. Rosenberg, "Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)," RFC 5766, RFC Editor, Fremont, CA, USA, pp. 1–67, Apr. 2010, updated by RFC 8155. [Online]. Available: https://www.rfc-editor.org/rfc/rfc5766.txt

[12] Z. Shelby, M. Koster, C. Bormann, P. V. der Stok, and C. Amsüss, "CoRE Resource Directory," Internet Engineering Task Force, Internet-Draft draft-ietf-core-resource-directory, Mar. 2019, work in Progress. [Online]. Available: https://datatracker.ietf.org/doc/html/draft-ietf-core-resource-directory-20

[13] D. Wing (Ed.), S. Cheshire, M. Boucadair, R. Penno, and P. Selkirk, "Port Control Protocol (PCP)," RFC 6887, RFC Editor, Fremont, CA, USA, pp. 1–88, Apr. 2013, updated by RFCs 7488, 7652, 7843. [Online]. Available: https://www.rfc-editor.org/rfc/rfc6887.txt

[14] P. Srisuresh and K. Egevang, "Traditional IP Network Address Translator (Traditional NAT)," RFC 3022, RFC Editor, Fremont, CA, USA, pp. 1–16, Jan. 2001. [Online]. Available: https://www.rfc-editor.org/rfc/rfc3022.txt