Islam, Hasan M.A.; Lagutin, Dmitrij; Fotiou, Nikos

Observing IoT resources over ICN

Published in:
2017 IFIP Networking Conference, IFIP Networking 2017 and Workshops

DOI:
10.23919/IFIPNetworking.2017.8264878

Published: 18/01/2018

Document Version
Peer-reviewed accepted author manuscript, also known as Final accepted manuscript or Post-print

Please cite the original version:
Islam, H. M. A., Lagutin, D., & Fotiou, N. (2018). Observing IoT resources over ICN. In *2017 IFIP Networking Conference, IFIP Networking 2017 and Workshops* (Vol. 2018-January, pp. 1-8). IEEE.
https://doi.org/10.23919/IFIPNetworking.2017.8264878

Observing IoT Resources over ICN

Hasan M A Islam, Dmitrij Lagutin
Department of Computer Science
Aalto University
Espoo, Finland
Email: firstname.lastname@aalto.fi

Nikos Fotiou
Department of Informatics
Athens University of Economics and Business
Athens, Greece
Email: fotiou@aueb.gr

***Abstract*—The Constrained Application Protocol (CoAP) is an HTTP-like protocol for resource-oriented applications intended to run on constrained devices, typically part of the Internet of Things. CoAP observe is an extension to the CoAP protocol that allows CoAP clients to observe a resource through a simple publish/subscribe mechanism. The resource is a network data object or service that can be identified by URI [1]. In this paper we leverage Information-Centric Networking (ICN), transparently deployed within the domain of a network provider, to provide enhanced CoAP services. We present the design and implementation details of the CoAP observe over ICN and discuss how ICN can provide benefits to both network providers and CoAP applications, even though the latter are not aware of the existence of ICN. In particular, the use of ICN results in smaller state management complexity at CoAP endpoints, simpler implementation at CoAP endpoints, and less communication overhead in the network.**

I. Introduction

The Internet of Things (IoT) is expected to interconnect billions of devices that will generate vast amounts of information. Significant efforts have been devoted into enabling smart devices to connect to the Internet, share information, and consume services. These efforts have resulted in a variety of network access technologies and higher layer protocols. On the other hand, core (inter-)networking technologies have not been adapted to this new paradigm, raising concerns about whether or not networks will be able to cope with the scale and the patterns of the traffic of the IoT. In order to assuage these concerns a number of researches have sprung up proposing Future Internet (FI) architectures. One such promising FI architecture is Information-Centric Networking (ICN). [1] ICN advocates implementing all (inter-)networking functions around content (i.e., information) identifiers, rather than location identifiers. This shift in focus and techniques is expected to overcome various limitations of the current Internet [3]. However, such a shift requires not only the redesign of networking protocols, but also the modification of legacy Internet applications. Such radical changes at all network layers are an overwhelming barrier to the adoption of ICN. With this in mind, the POINT project [4] proposes a radical approach to ICN adoption: it postulates an individual ICN operator that uses network attachment points that translate *legacy* IP applications traffic to ICN, i.e., the endpoints are oblivious to ICN.

The Constrained RESTful Environments (CoRE) working group has designed and developed Constraint Application Protocol (CoAP) [5] which is intended to operate in constrained IP networks and provides RESTful services in constrained devices. The CoAP interaction model is similar to the client/server model of HTTP: a CoAP client issues a request message to a server and if the CoAP server is able to serve the request, it responds to the requester with a response code and the payload. Unlike HTTP, CoAP requests and responses are exchanged asynchronously, on top of an unreliable datagram oriented transport protocol (e.g., UDP).

CoAP observe is described in [6] as an extension to the CoAP protocol. CoAP observe enables client to observe a resource through a simple publish/subscribe mechanism. The server asynchronously pushes the notification of state changes of the resource for which the client is interested in and follows a best-effort approach to guarantee the eventual consistency of the observed state and the actual state of the resource. Compared to HTTP, observe extension to CoAP protocol can significantly reduce communication overhead in terms of bandwidth requirements and number of messages transmitted. Since CoAP observe protocol is based on publish/subscribe paradigm, it can benefit from The POINT architecture in terms of latency, state management, communication overhead and better security and privacy.

In this paper, we present the design and implementation of CoAP observe over ICN and describe how CoAP can benefit from the POINT architecture as well as, how a network operator that offers CoAP connectivity can benefit from ICN. Moreover, we show that by leveraging the multicast nature of the POINT, an operator can achieve significant benefits in terms of communication overhead. It is important to note that our solution enables the usage of *legacy IP-based devices*, for instance, the existing CoAP endpoints can transparently access the resources hosted in IoT devices through the intermediate ICN network.

The remainder of this paper is organized as follows. In Section II we describe the CoAP protocol, as well as, the information-centric POINT architecture. Section III presents the main motivation towards this work. In Section IV we illustrate CoAP observe over ICN reference architecture, alongside with the implementation details. Section IV also highlights

[1] A survey of ICN research and architectures that have been investigated, with some still being pursued and experimentally explored further, can be found in [2].

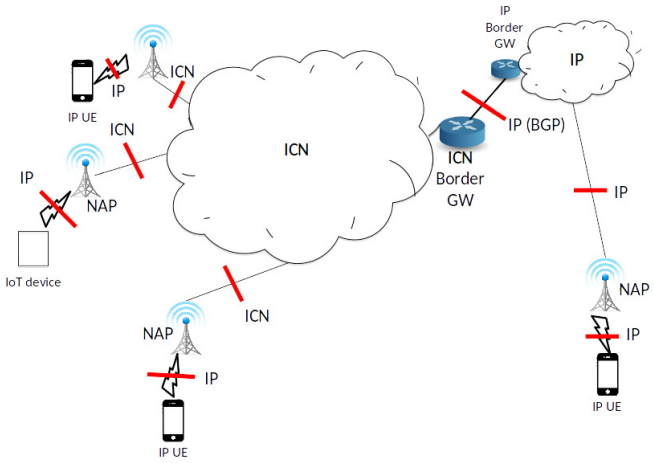


Fig. 1: The POINT architecture.

how ICN can benefit CoAP-based applications. In Section V we present the node operations to support CoAP observe in ICN architecture and how ICN can benefit to CoAP endpoints. Section VI explores the related efforts and finally, Section VII concludes our paper.

II. Background

A. CoAP Observe

The CoAP protocol supports intermediaries and caching of responses. The intermediaries or proxies perform the role on behalf of the CoAP endpoints. For instance, a forward proxy acts on behalf of the CoAP client and sends a request to the CoAP server. The forward proxy needs to be explicitly configured to perform requests on behalf of the client. On the other hand, a reverse proxy is transparent to the client. The reverse proxy performs on behalf of the CoAP server and behaves as if it were the origin server. CoAP observe protocol supports to observe a resource through CoAP proxy. The server asynchronously pushes the notification of state changes of the resource for which the client is interested in and follows a best-effort approach to guarantee the eventual consistency of the observed state and the actual state of the resource. To achieve this, a CoAP client needs to register with a CoAP server using the GET request with the observe option. The CoAP client can also utilize a proxy for observing a resource.

Recent efforts [7], [8] have been performed on proxy based CoAP observe in Wireless Sensor Network (WSNs). Alessandro et el. [7] includes WebSocket protocol in the design of the CoAP proxy for HTTP based web applications. The work in [8] considers dynamic aggregation/scheduling of multiple observe requests at CoAP proxies. These efforts are complementary to our work, which utilizes ICN to further enhance the efficiency gains of the CoAP observe.

B. The POINT architecture

Instead of dictating a clean-slate end-to-end ICN architecture, which would be very challenging to deploy, POINT allows standard IP traffic to be run over an ICN core network in a more efficient way [4]. To achieve this, the POINT architecture (Figure 1) provides a number of handlers implemented by the Network Attachment Points (NAPs). These handlers perform translations between the existing IP-based protocols (e.g., HTTP, CoAP, basic IP) and appropriate named objects within the ICN core on both edges of the core. Therefore existing applications can benefit from ICN's features such as native multicast and caching without any modifications. The potential benefits of the POINT architecture compared to IP-based ones are highlighted in more detail in [4] (i) better utilisation in HTTP unicast streaming scenarios, (ii) better security and privacy for constrained applications, (iii) better management of virtualized network paths, (iv) better (fairer) content distribution.

In the POINT ICN architecture, content is organized using *scopes* with an identification structure that provides hierarchically organized information [9]. Every content item belongs to at least one scope which is identified by a Scope Identifier (SId). The purpose of a scope is to give a hint about content location and to group content items with the same dissemination level. The POINT architecture decomposes the core network functions into three parts: scope specific *Rendezvous*, *Topology* and *Forwarding* (RTF) functions. The rendezvous network provides a lookup service, which routes a subscription to a RN that "knows" (at least) one publisher for the requested item. Scopes are managed by specialized Rendezvous Nodes (RNs), which form an overlay Rendezvous Network. The rendezvous network provides a lookup service, which routes a subscription to a RN that "knows" (at least) one publisher for the requested item. To publish a new content, publishers assigns two identifiers: a unique label for every piece of content referred as *rendezvous identifiers* RId and scope identifiers (SId). Upon receiving a subscription message and provided that at least one publisher exists, the RN instructs a *Topology Manager* to create a forwarding path from a publisher to the subscriber. The forwarding function utilizes *Forwarding identifiers* (FId) that denote the forwarding tree as a form of a source routing [10]. Each node along the path maintains a forwarding table that maps the incoming FId to corresponding outgoing interfaces. Finally, the content item is transferred from the publisher to the subscriber.

III. Motivation

The Internet of Things is a recent paradigm that interconnects billions of heterogeneous devices, ranging from wireless sensors to actuators, wearable devices, Radio-Frequency Identification (RFID) tags, home appliances, surveillance cameras and many other types of machines. It is expected that these devices are uniquely identified and can communicate with each other. The Internet Engineering Task Force (IETF) has developed a suite of protocols and open standards for IoT. However, the design of a networking architecture for IoT devices, which may be resource-constrained and also the traffic patterns are highly heterogeneous, poses great challenges [11]. First, there is a need of additional resolution systems

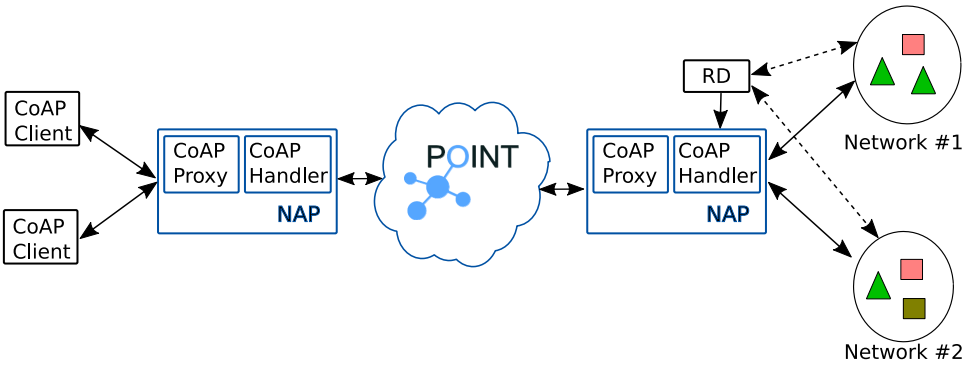


Fig. 2: An example of CoAP over ICN reference architecture. On the right part there are Things offering resources. Each resource is specified by a color and also by shape. On the left part there are CoAP clients.

to translate resource URI into IP addresses. However, it is difficult for constrained devices to allocate more resources for the DNS client implementation. Second, some of the constrained devices (e.g., temperature sensor) may receive a vast amount of requests from clients which require significantly high processing loads. CoAP supports caching and proxy that can reduce the access traffic from the client that have already issued the CoAP requests but unable to get rid of the CPU loads caused by the new clients. Third, CoAP supports the CoAP observe, where a client can subscribe to a particular resource and the CoAP server responds when the status of the resource changes. However, the CoAP server needs to stay awake to serve the possible requests from the clients. Furthermore, the CoAP server needs to maintain state for each client and responds separately for each client. In this paper, we address these limitations and provide solution to reduce the access to a CoAP server for a particular resource.

IV. CoAP Observe over ICN

In this section we discuss the design and implementation of the CoAP observe over ICN. This module is a part of our earlier CoAP over ICN work [12] that illustrates various CoAP-specific communication scenarios.

Figure 2 illustrates the network setup for observing resources over the ICN architecture. In the middle of the figure there is the POINT network that interconnects NAPs. In the right part of the figure there are networks of *Things*. Each Thing acts as a CoAP server offering a resource; the same (type of) resource can be offered by many Things located in different networks (e.g., there can be many sensors deployed in various parts of a city offering temperature measurements). Each network of Things is connected to the POINT network through a NAP. A network of Things may be directly attached to a NAP. A CoAP Resource Directory (RD) hosts the descriptions of resources provided by the CoAP servers. In the left part of the figure there are CoAP clients. A CoAP client is also connected to the POINT network though a NAP.

The fundamental component of our CoAP over ICN architecture is a *CoAP handler* which is part of the NAP. A CoAP handler receives CoAP requests from CoAP clients (over IP), performs protocol translation and forwards the requests to CoAP server. The CoAP server generates a response which is forwarded through the ICN network to the CoAP clients following the reverse process.

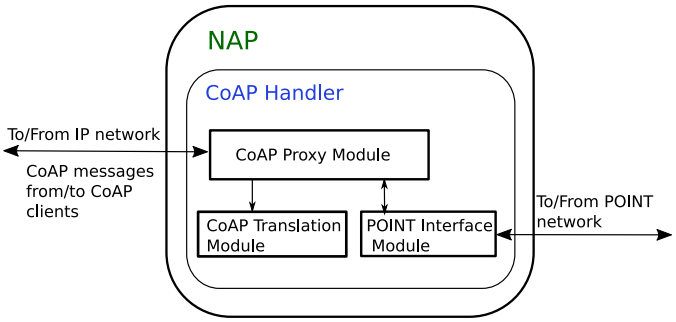


Fig. 3: CoAP handler functional module.

A. *Functional Requirements*

To implement CoAP observe over ICN, CoAP handler has the following functional requirements as shown in Figure 3:

- The CoAP protocol runs on top of UDP. The CoAP handler MUST maintain states of the CoAP clients prior forwarding the request to the POINT network. This allows the CoAP handler to forward the corresponding CoAP response back to the appropriate client.
- Efficient maintenance of the state within the CoAP handler in case of resource subscriptions. The CoAP handler should maintain some states when similar requests are issued by multiple clients attached to the same client-side NAP (cNAP). Similarly, server-side NAP (sNAP) should also consider how to efficiently handle multiple requests for the same resource from multiple cNAPs.
- The CoAP handler MUST follow the protocol semantics of CoAP observe [6] when registering a CoAP client to the CoAP server.

B. *Functional Modules*

CoAP handler is composed of the following functional module modules:

Proxy: The CoAP handler can be classified as one of the two categories depending on the role it provides to the POINT architecture; forward proxy when it performs requests on behalf of the client and reverse proxy when it behaves as if it were the original server. In our design, both roles are combined in the same module and are complementary.

Protocol Translation: This module allows the CoAP handler to translate the CoAP messages to ICN messages and vice versa. The translation module follows the semantics of CoAP protocol (RFC 7252).

POINT Interface Module This module advertises ICN messages (translated from CoAP requests) to the ICN network. The identifier of these messages includes the host-URI of the CoAP resource, therefore they trigger the ICN rendezvous process, which eventually leads to forwarding of these messages to the appropriate NAP(s) on the other side of the ICN network. A NAP that receives an ICN message restores the original CoAP request and forwards it to the appropriate CoAP server. The CoAP server generates a response which is forwarded through the ICN core network back to the CoAP clients following the reverse process.

Algorithm 1 Handling GET request for CoAP observe

1: $isObserveOption \leftarrow [coap\_request]$
2: $token \leftarrow [coap\_request]$
3: $messageID \leftarrow [coap\_request]$
4: $resourceURI \leftarrow [coap\_request]$
5: **if** $isObserveOption == true$ **then**
6: $search\_subscription\_list(tokenFlag, uriFlag)$
7: **if** $tokenFlag == true$ **then**
8: $drop\ the\ request\ packet$
9: **end if**
10: **if** $uriFlag == true\ \&\&\ tokenFlag \neq true$ **then**
11: $insert\ an\ entry\ in\ subscription\_list$
12: **else**
13: $create\ coap\ request\ with\ observe\ option$
14: **end if**
15: **end if**

V. Node Operation

A. Module Overview

The proxy module of the CoAP handler provides the core functionality of the CoAP proxy. The translation process of CoAP messages to ICN messages that follows the semantics of CoAP protocol [5] is also implemented in this module. The CoAP request message and the information related to it are stored in the `client_node` structure. Since CoAP transport is based on UDP, the `client_node` allows the CoAP handler to match the response received by the CoAP device with the request it has sent and forwards the response to the appropriate CoAP client. CoAP proxy module implements a list of `client_node` to keep track of pending requests and observing the resources.

```
struct client_node {
  struct client_node* next;
  struct sockaddr_storage addr;
  socklen_t addr_len;
  unsigned char * token;
};
```

The CoAP handler receives the CoAP request message, if the NAP is configured as a proxy. The proxy module of the CoAP handler translates the CoAP request and extracts the first 4-byte mandatory header, which contains the basic information of the CoAP request, such as CoAP message type, method code (GET, PUT, DELETE, UPDATE) and token length. The value of the token length indicates the presence of the Token value. The request URI of the resource in a proxy request is specified as a string in the PROXY-URI option. The request also includes Token which is used for matching responses with requests and aggregate similar requests in both cNAP and sNAP. The request URI of the resource, for which the client is interested in to an origin server, is split into the URI-HOST, URI-PORT, URI-PATH, and URI-QUERY Options. The URI-HOST is the FQDN of the CoAP server and the URI-PATH is the path of the resource within the server.

The URL of the desired resource is obtained by concatenating the URI-Host and the URI-Path options. Afterwards the proxy constructs a new CoAP request. This request uses the 4-byte mandatory header of the original request and includes all the splitted options from PROXY-URI option. The request also includes Token in the new request.

B. Handling Observe Request

Algorithm 3 illustrates the operation to process a observe request in the CoAP handler. Processing observe request is performed only if the request message contains the CoAP observe option. The first step of processing observe request is intended to establish an observe relationship between the CoAP client and the CoAP server. To achieve this, the proxy module maintains the subscription list. The structure of the subscription list is as follows:

```
struct coap_subscription{
    char* resource_uri;
    int resource_uri_len;
    unsigned char* token;
    int token_len;
    int iteration;
    uint16_t message_id;
    struct coap_subscription* next;
    struct client_node* client;
  };
```

First, the module checks the `coap_subscription` list to verify if the request is retransmitted by the client. If the

match is found, it drops the packet. Second, the module verifies whether if the request message is intended for the resource which already exists in the `coap_subscription` list. If the match is found, it inserts a new entry in the `coap_subscription` list for this request. The proxy module sends the observe request only if no entry exists in the `coap_subscription` list with the same resource. The CoAP handler of the NAP is the only observer registered directly to the CoAP server and distributes the resource information to CoAP clients. In turn, our solution reduces the resource consumption of the CoAP device and communication overhead.

Algorithm 2 Handling coap response from coap server

```
1: isObserveOption ← [coap_response]
2: token ← [coap_response]
3: if isObserveOption = = true then
4:    found ← search_observer(token, resource_uri)
5:    observer_entry ← subscription_list
6:    while observer_entry ≠ NULL && observer_entry → resourceURI
      == resource_uri do
7:       token ← observer_entry.token
8:       messageID ← observer_entry.messageID)
9:       iteration ← observer_entry.firstResponseFlag
10:      if iteration then
11:         #To create observe relationship
12:         insert ACK code in response
13:      end if
14:      observer ← observer_entry.client
15:      update response with token and messageID
16:      send_response(coap_response, observe)
17:      next observer_entry
18:   end while
19: end if
```

C. Handling Observe Response

Algorithm 2 illustrates the operation to process the response packet of the observe request. The CoAP handler checks the response packet to verify if the the response packet contains the CoAP observe option. If Observe option is found, the proxy follows the Algorithm 2 to process the response packet. The response packet only echoes the token and message identifier and does not have any information about the request URI for this response. Therefore, the proxy module first checks the `coap_subscription` list to find a match. If the match is found, it extracts the resource URI information for this response. Based on the resource URI information, the proxy collects all the observer informations and distribute the response to the observers. The `coap_subscription` list also provides information if this is the first response for any observer. This is necessary in a case when multiple CoAP clients are interested in the same resource for which an observe relationship has already established with a CoAP client. For subsequent observe relationship, the proxy checks the value of `iteration`. If the value is zero, the CoAP handler updates the message status code with Acknowledgement and inserts the appropriate message id and sends the response to every interested observer for the similar resource. The handler also change the `iteration` value to 1. When the CoAP client receives the CoAP response for the observe request, it verifies the token and message id. If the verification is successful the observe relationship will be established.

D. Handling Acknowledgement

When a CoAP client issues observe request, it waits for an acknowledgement from the CoAP server. Upon reception of the acknowledgement, an observe relationship between a CoAP client and CoAP server is established. The CoAP client also echoes 4 byte mandatory header to notify the CoAP server that the client is alive and interested in receiving further notifications. This procedure is intelligently handled by the CoAP handler in case of multiple observe requests for the similar resource hosted in a CoAP server. To achieve this, the proxy module of the CoAP handler maintains the list of pending acknowledgements for those requests which are not forwarded to CoAP server. The structure of the list is as follows:

```
struct suppressed_token {
 unsigned char * token;
 int token_len;
 uint16_t message_id;
 struct suppressed_token* next;
};
```

Algorithm 3 Handling ACK

```
1: message_id ← [coap_ack]
2: if message_id ∈ suppressed_token then
3:    do nothing
4: else
5:    send coap_ack to coap server
6: end if
```

Note that the acknowledgement packet from the CoAP clients contains only 4-byte mandatory header of CoAP protocol, which echoes the message identifier but does not include the token value.

E. Message Sequence Diagram

A communication (CoAP request) with an CoAP server is a publication to its fully qualified domain name (FQDN), represented as Content Identifier (CID), while the CoAP responses are delivered to the URL, hashed over a carefully selected set of request parameters, of which the URL is only one. This response is represented as reverse CID (rCID). A request from a CoAP client of any device is interpreted as an appropriate ICN name which can be related to the FQDN of the server, while the response to that request is published to the appropriate ICN name related to the URL of the request. This allows a server NAP to simply subscribe to the FQDN of

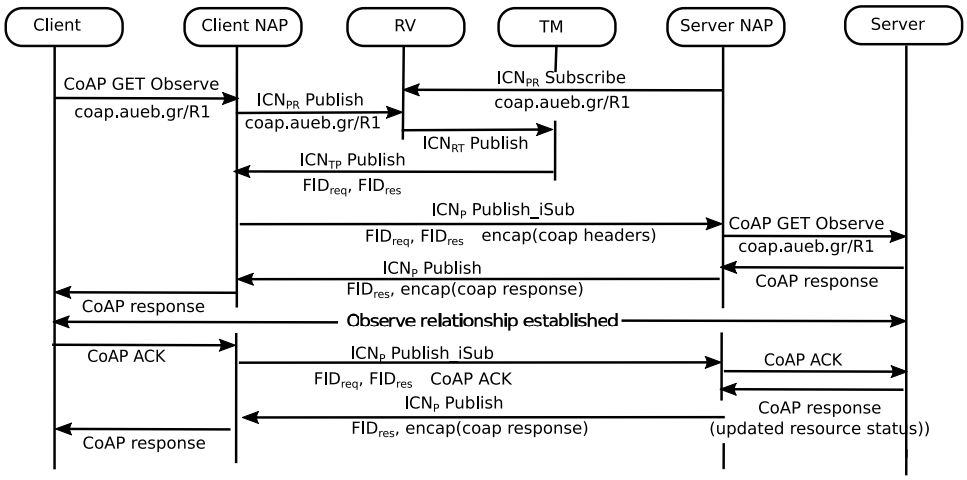


Fig. 4: Message Sequence Diagram from a CoAP client to CoAP server.

any attached CoAP server, while the CoAP server can publish any response to the corresponding URL.

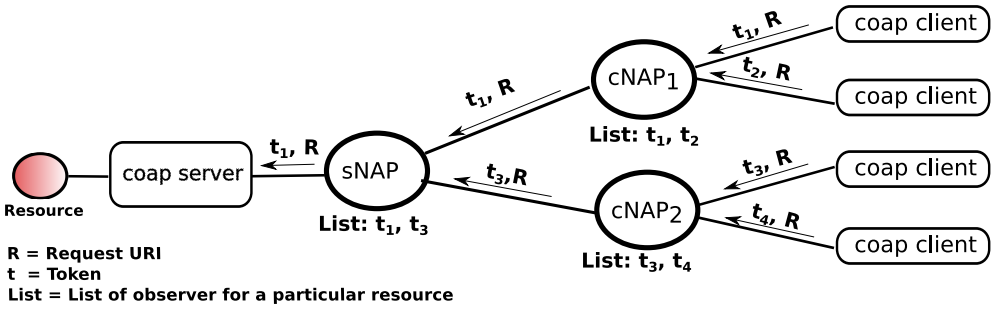
In Figure 4, we show an example of message sequence chart (MSC) for a CoAP client request and CoAP server response. The CoAP server registers its DNS name which leads to its NAP subscribing to the server's FQDN (aueb.example.gr/R1). CoAP client issues a CoAP observe request for the resource aueb.example.gr/R1. cNAP receives it and publishes to the Rendezvous (RVZ). the cNAP uses the pub_isub API (publish with implicit subscription) call to publish the CoAP request. The RVZ matches it with the servers subscription and asks the TM (as part of its internal realization) to create a forwarding path for the request as well as the reverse path. Eventually the TM sends the forwarding path (FID$_{req}$) and reverse path (FID$_{res}$) to the cNAP. cNAP forwards the request to sNAP using (FID$_{req}$). sNAP de-capsulate the CoAP request and sends it to the CoAP server. The CoAP server processes the request and sends the CoAP observe response to sNAP. sNAP forwards it to cNAP using (FID$_{res}$). Finally cNAP forward the response to appropriate client. Upon reception of the CoAP response, the observe relationship between the CoAP client and CoAP server is established. The subsequent communication from the CoAP client to transport ACK and the updated resource status is performed using (FID$_{req}$) and (FID$_{res}$).

F. Benefits

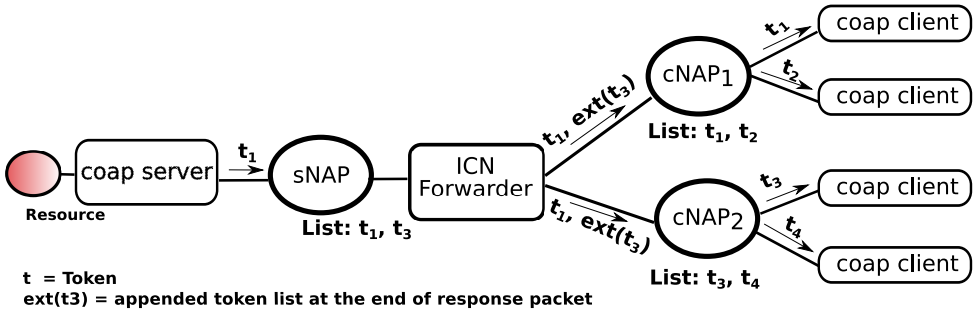
We now present a simple CoAP communication scenario of observing a resource in Figure 5 and discuss the benefits of the ICN underlay to CoAP. In Figure 5a CoAP clients issue observe requests for a resource **R**, including token **"t1", "t2", "t3", "t4"** respectively. cNAP$_1$ receives CoAP observe requests from a CoAP client with **"t1"**. cNAP$_1$ forwards the request to sNAP and creates an entry in the subscription list. sNAP receives the request and creates an entry in the subscription list for cNAP$_1$ and sends the request to the CoAP server. If the observe relationship is possible, the CoAP server replies with ACK which eventually creates an observe relationship between CoAP client and CoAP server through NAPs. After some time, another CoAP client issues observe request to cNAP$_1$ with Token **"t2"**. cNAP$_1$ finds a match for this request and insert a new entry in the subscription list and does not forward the request to sNAP. Similarly, cNAP$_2$ receives two requests with **"t3"**, **"t4"** and forward the request to sNAP with **"t3"**. sNAP receives the request from cNAP$_2$. sNAP finds a match and inserts a new entry for cNAP$_2$. cNAP and sNAP only forward the request which arrives earlier and maintain the list for others. The communication between cNAP and sNAP uses ICN message which is translated from the CoAP message.

The main benefit to use the POINT architecture is exploiting the native multicast of ICN architecture. Figure 5b shows that CoAP server sends one unicast response to sNAP. sNAP translates it into an ICN message and sends it to cNAP$_1$ and cNAP$_2$. Finally cNAP$_1$ and cNAP$_2$ forward the response to appropriate client including the correct tokens and message identifiers. In typical IP networks this communication pattern would result in multiple unicast transmissions from the CoAP server to the CoAP clients. In contrast, in POINT the impact to the network of this type of bursty traffic can be reduced by employing multicast. In order to achieve this, the CoAP handler instructs NAPs to use the same token for all these identical CoAP requests. NAPs are then responsible for modifying the token to the CoAP response.

Furthermore, by using ICN, we will be able to provide better security and privacy for the constrained applications. Let us consider the example of a multi-tenant building, where various sensors have been deployed. The building management system includes the energy monitoring (e.g., temperature and humidity measurements), the security and safety of the building (e.g., motion detection, fire alarms), billing (e.g., energy consumption, number of parking slots used), and so on. It is expected that each tenant should be able to define access control policies since the information provided to building management system is very sensitive. However, it is



(a) CoAP observe request.



(b) CoAP observe response.

Fig. 5: A simple scenario of observing a resource through POINT platform. (a) multiple observe requests for the similar resource **R** hosted in a CoAP server (b) Forwarding the response to CoAP clients.

not feasible to extend the constrained device to support the access control policies, both from the performance/cost and security perspectives, because it will increase the processing power requirements and energy consumption. Moreover, it will expose the sensitive information (e.g., user credentials) to many entities. Nonetheless, the NAP is able to collect all information and implement information access policies. The CoAP handler is able to associate security and privacy requirements with namespaces, enabling the definition of fine-grained, reusable access rules that will govern information access directly from personal gateways.

G. Evaluation

We tested our design with two clients, one forward proxy, one reverse proxy and one CoAP server. The CoAP server and clients runs libcoap[2]. Forward and Reverse proxy runs our software. Both CoAP clients issue observe requests for the same resource containing different tokens. The duration of observing the resource is 90s. This duration is default timeout in libcoap. The evaluation shows that 48% of the total request packets including acknowledgements are forwarded to CoAP server. The maximum transmission is performed between CoAP clients and forward proxy. The communication overhead in bytes is reduced by 50%. We note that the performance is a function of the number of clients issuing similar requests.

We will integrate our solution in the POINT architecture. To evaluate the performance of CoAP observe over ICN, we will construct a simple IoT testbed which will be connected to the existing POINT testbed through NAP. The POINT testbed connects all partners of the POINT project throughout Europe using OpenVPN. The POINT overlay testbed is based on Blackadder ICN platform, which has already demonstrated ICN performance at data rates up to 10 Gb/s [13].

[2]https://github.com/obgm/libcoap

VI. Related Work

In recent years, several networking architectures have been proposed for Futrue Internet (e.g., CCN [14], DONA [15], PURSUIT/PSIRP [16], POINT [4], NetInf [17] with aspiration to efficiently distribute and retrieve content. However, those architectures may not be directly applicable in the IoT due to different requirements imposed by the constrained devices in terms of memory, power capacity and processing computational loads. These limitations have attracted research community to study ICN paradigms in IoT area. In [18], the authors highlight the key challenges of IoT and provide a design of high level NDN architecture that can meet IoT challenges. In [19], the authors propose the CCN communication layer on top of MAC layer to transmit packets. Similarly, an overlay of ICN architecture based on CCN on top of ETSI M2M architecture is presented in [20]. In [21], authors propose a service platform based on CCN for Smart Cities that can integrate the available relevant wireless technologies to provide ubiquitous services, optimize the usage of communication resources through distributed caching and provide security by exploiting the security feature of CCN architecture. In [22], the authors provide an experimental comparison of CCN with the traditional IP based IoT standards

6LoWPAN/RPL/UDP in terms of energy consumption and memory footprint. This experiment has used a compact version of CCNx [23], referred as CCN-Lite [24], in RIOT OS [25]. The authors of [26] propose a push mechanism for CCN to optimize the traffic in sensor networks, whereas authors of [27] propose a content-centric internetworking scheme for resource constrained network devices based on task mapping where the netork activities (e.g., storing, publishing, and retrieving content) of the constrained devices are transferred to the core CCN network.

In contrast, the POINT architecture provides CoAP handler to map CoAP protocol onto appropriate named objects within the ICN core. The CoAP endpoints are connected through ICN using the NAP. The CoAP handler is part of the NAP. This handler provides CoAP services to the CoAP endpoints using ICN. The CoAP handler of cNAP/sNAP is responsible to manage the appropriate states for the CoAP requests and reduces the access to the CoAP server for the similar requests targeted to a particular resource. Furthermore, the NAP transparently establishes the observe relationship between a CoAP server and a CoAP client. The NAP also suppresses ACK packets of those clients who later request the same resource and there has already been an established observe relationship between the CoAP server and client.

VII. Conclusions and Future work

In this paper, we present the design and implementation details of CoAP observe for POINT architecture that enable CoAP clients to observe the resources hosted in IoT devices through ICN network. The CoAP clients are oblivious to the existence of ICN. CoAP observe functionality is very similar to publish/subscribe based ICN in particular asynchronous transmission. Transporting CoAP traffic over ICN can benefit in terms of communication overhead, state management and latency, in particular when multiple clients are interested to subscribe the same resource hosted in a IoT device. In addition, the inherent multicast capabilities of ICN and caching at the edge can be exploited in observing similar resources hosted in IoT devices by multiple CoAP clients.

Acknowledgments

Many of the ideas presented in this paper stem from discussions among POINT consortium partners. The work presented in this paper was supported by the EU funded H2020 ICT project POINT, under contract 643990

References

[1] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Rfc 2616-http/1.1, the hypertext transfer protocol," 1999.

[2] G. Xylomenos, C. Ververidis, V. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. Katsaros, and G. Polyzos, "A Survey of Information-Centric Networking Research," *IEEE Communications Surveys Tutorials*, vol. 16, no. 2, pp. 1024–1049, 2014.

[3] D. Trossen, M. Sarela, and K. Sollins, "Arguments for an information-centric internetworking architecture," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 2, pp. 26–33, Apr. 2010.

[4] D. Trossen, M. J. Reed, J. Riihijrvi, M. Georgiades, N. Fotiou, and G. Xylomenos, "IP over ICN - The better IP?" in *European Conference on Networks and Communications (EuCNC),*, June 2015, pp. 413–417.

[5] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (CoAP)," IETF, RFC 7252, 2014.

[6] K. Hartke, "Observing resources in the constrained application protocol (CoAP)," IETF, RFC 7641, 2015.

[7] A. Ludovici and A. Calveras, "A proxy design to leverage the interconnection of coap wireless sensor networks with web applications," *Sensors*, vol. 15, no. 1, pp. 1217–1244, 2015.

[8] N. Correia, D. Sacramento, and G. Schutz, "Dynamic aggregation and scheduling in coAP/observe based wireless sensor networks," *IEEE Internet of Things Journal*, 2016.

[9] N. Fotiou, D. Trossen, and G. C. Polyzos, "Illustrating a publish-subscribe internet architecture," *Telecommunication Systems*, vol. 51, no. 4, pp. 233–245, 2012.

[10] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander, "LIPSIN: line speed publish/subscribe inter-networking," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 195–206, Aug. 2009.

[11] Z. Sheng, S. Yang, Y. Yu, A. Vasilakos, J. Mccann, and K. Leung, "A survey on the IETF protocol suite for the internet of things: Standards, challenges, and opportunities," *IEEE Wireless Communications*, vol. 20, no. 6, pp. 91–98, 2013.

[12] N. Fotiou, H. Islam, D. Lagutin, T. Hakala, and G. C. Polyzos, "CoAP over ICN," in *2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, Nov 2016, pp. 1–4.

[13] J. Riihijärvi *et al.*, "Final architecture validation and performance evaluation report," *EU FP7 PURSUIT Deliverable D*, vol. 4, p. 5, 2012.

[14] V. Jacobson, D. Smetters, J. Thornton, M. Plass, N. Briggs, and R. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 1–12.

[15] T. Koponen, M. Chawla, B. Chun, A. Ermolinskiy, K. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," in *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4. ACM, 2007, pp. 181–192.

[16] S. Tarkoma, M. Ain, and K. Visala, "The publish/subscribe internet routing paradigm (PSIRP): Designing the future internet architecture," *Towards the Future Internet*, p. 102, 2009.

[17] C. Dannewitz, "Netinf: An information-centric design for the future internet," in *Proc. 3rd GI/ITG KuVS Workshop on The Future Internet*, 2009.

[18] M. Amadeo, C. Campolo, A. Iera, and A. Molinaro, "Named data networking for iot: An architectural perspective," in *Networks and Communications (EuCNC), 2014 European Conference on*. IEEE, 2014, pp. 1–5.

[19] B. Saadallah, A. Lahmadi, and O. Festor, "Ccnx for contiki: implementation details," Ph.D. dissertation, INRIA, 2012.

[20] L. A. Grieco, M. B. Alaya, T. Monteil, and K. Drira, "Architecting information centric etsi-m2m systems," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2014 IEEE International Conference on*. IEEE, 2014, pp. 211–214.

[21] I. Cianci, G. Piro, L. A. Grieco, G. Boggia, and P. Camarda, "Content centric services in smart cities," in *Next Generation Mobile Applications, Services and Technologies (NGMAST), 2012 6th International Conference on*. IEEE, 2012, pp. 187–192.

[22] E. Baccelli, C. Mehlis, O. Hahm, T. C. Schmidt, and M. Wählisch, "Information centric networking in the iot: Experiments with ndn in the wild," in *Proceedings of the 1st international conference on Information-centric networking*. ACM, 2014, pp. 77–86.

[23] P. CCNx, "http://www.ccnx.org," Sep. 2009.

[24] C. Lite, "Lightweight implementation of the content centric networking protocol," 2014.

[25] E. Baccelli, O. Hahm, M. Gunes, M. Wahlisch, and T. C. Schmidt, "Riot os: Towards an os for the internet of things," in *Computer Communications Workshops (INFOCOM WKSHPS), 2013 IEEE Conference on*. IEEE, 2013, pp. 79–80.

[26] J. François, T. Cholez, and T. Engel, "Ccn traffic optimization for iot," in *Network of the Future (NOF), 2013 Fourth International Conference on the*. IEEE, 2013, pp. 1–5.

[27] Y. Song, H. Ma, and L. Liu, "Content-centric internetworking for resource-constrained devices in the internet of things," in *Communications (ICC), 2013 IEEE International Conference on*. IEEE, 2013, pp. 1742–1747.