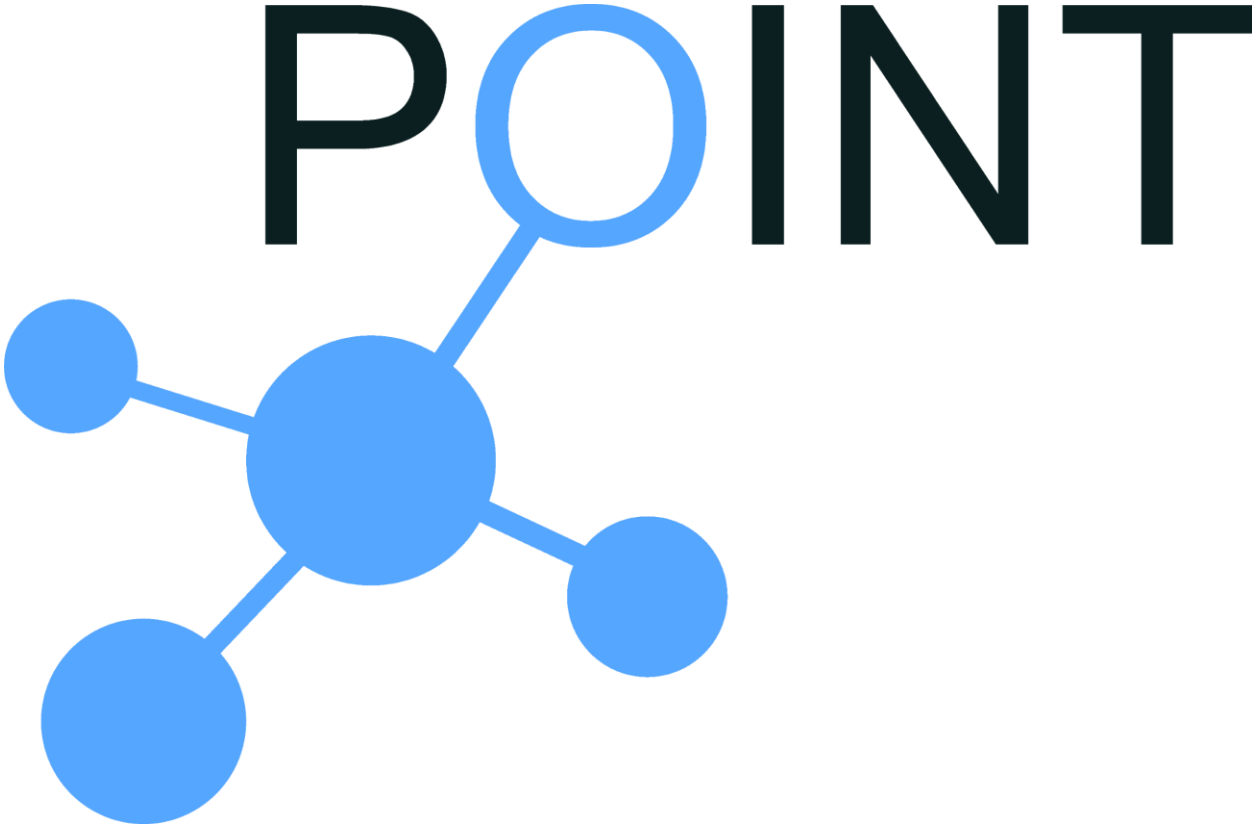
H2020 i**P** **O**ver **I**c**N**- the bet**T**er IP (POINT)

# Design-CoAP

## POINT CoAP Handler Design




**List of Authors:** Dmitrij Lagutin, Nikos Fotiou

1. Architecture

2. CoAP Handler

2.1 Message Sequence Example

# 1. Architecture

CoAP is based on RESTful principles, as is HTTP. CoAP has been designed and developed to be a 'lightweight HTTP' so that it can operate in constrained IP networks and devices. The CoAP interaction model is similar to the client/server model of HTTP. A CoAP client can issue a request message to a server using a method code on a resource, which is identified by a URI scheme. 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). Furthermore, CoAP offers some salient features for constrained environments, such as discovery of resources and (unreliable) multicast.

CoAP operates under a simple framework, consisting of the following two layers on top of UDP: the Messaging layer and the Request/Response layer. The messaging layer handles the asynchronous nature of communication, while the request/response layer deals with request/response messages using Method and Response codes. The messaging model supports 4 types of messages: CON (confirmable), NON (non-confirmable), ACK (Acknowledgement), RST (Reset). Every CoAP message may carry a Token whose value is a sequence of 0 to 8 bytes. The Token correlates a response with a request, along with the additional address information of the corresponding CoAP endpoint. A CoAP client generates the Token for a request message and the server uses the same Token in the response. Each message also contains a 16-bit message ID, which is used to detect message duplicates.

The CoAP protocol also supports intermediaries and caching of responses. The intermediary or proxy is a crucial entity in the constrained network, since the proxy enables the constrained network to communicate with the global Internet. There are two different kinds of proxies: Forward-Proxy and Reverse-Proxy. A Forward-Proxy sends a request to the server on behalf of a client. For this, the Forward-Proxy needs to be configured to perform requests on behalf of the client. The server indeed treats the Forward-Proxy as a client. In contrast, a Reverse-Proxy is transparent to the client, behaving as if it were the origin server. Another important component of a CoAP-based system is the Resource Directory (RD). A RD acts similarly to a Web-based directory of resources provided by a HTTP server. Similarly, a RD provides a repository of links to resources hosted on other CoAP servers, with a lookup interface allowing clients to search available resources.

In many scenarios, the state of the resources may change over time. The CoAP core is not suitable for situations where a client is interested in the (current) state of a resource over a period of time. To achieve this, the CoAP core is extended with a new mechanism that enables clients to observe a resource (RFC 7641). With this, a CoAP client registers its interest in a resource using a GET request with the “Observe option". If the CoAP server accepts the request, the client receives an asynchronous notification message upon changes to the state of the resource.

Another salient feature that CoAP supports is Group communication (RFC 7390). The underlying mechanism of CoAP Group Communication is sending a single CoAP message to a specific group of devices, by exploiting UDP/IP multicast for the requests and unicast UDP/IP

for the responses. CoAP groups and the membership of a group can be discovered via the lookup interfaces in the Resource Directory (RD). The group membership of a CoAP endpoint can be pre-configured before deployment, or programmed to discover its group membership using a specific service discovery technique, such as DNS-based Service Discovery.

Figure 1 illustrates our CoAP over ICN architecture. CoAP clients interact with this architecture through a CoAP proxy implemented in the NAP. The proxy then uses the CoAP handler to transport CoAP messages over ICN. At the other part of the network, a CoAP helper is in charge of communicating with CoAP server endpoints and notifies CoAP handler when the response is available.

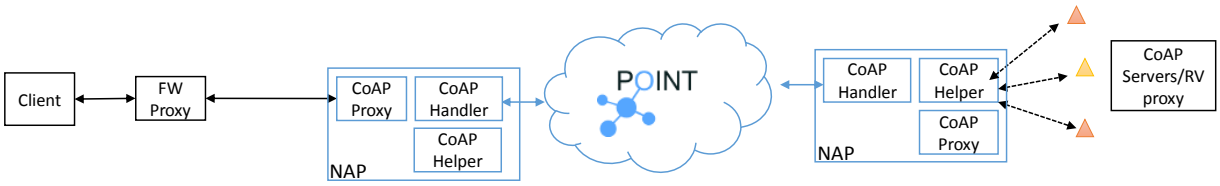



Figure 1: CoAP over ICN architecture

## 2. CoAP Handler

The CoAP standard and its extensions define various types of messages and a rich messaging model. The POINT CoAP handler supports confirmable GET and PUT requests, sent using UDP to a NAP acting as a forward proxy, separate (un)confirmable responses, interest registration/deregistration and resource updates for observable resources, and group communication. An essential part of the POINT CoAP handler is its exploitation of the native multicast capabilities of the POINT network in both directions: one client communicating with multiple servers and one server responding to multiple clients. Therefore we first explain how groups are formed and named, and then show server and group names are treated by the CoAP handler.

A CoAP server is identified by a Uri-Host and it may be part of multiple Groups. CoAP groups are used to organize CoAP servers that have some common attributes, e.g., they reside in the same location. We assume that these attributes are hierarchically organized, and create group names by assigning values to these attributes. In order to illustrate this concept, we consider a building management use case, depicted in Figure 2.

In this scenario, CoAP servers are located inside buildings and each server is attached to a NAP. In the presented scenario, buildings are numbered with a building number and then subdivided in wings and floors; these are the possible group attributes, which are hierarchically organized as shown in the left part of Figure 2. A group name is created by assigning “values” to (some of) the specified attributes, e.g., by setting building=building6, wing=west, and floor=floor3, the group name floor3.west.building6 is constructed. By omitting some attributes, we can create more general groups, for example, west.building6 or floor3.building6. From the CoAP handler’s point of view, both CoAP Uri-Host and CoAP group name are treated in the same way.

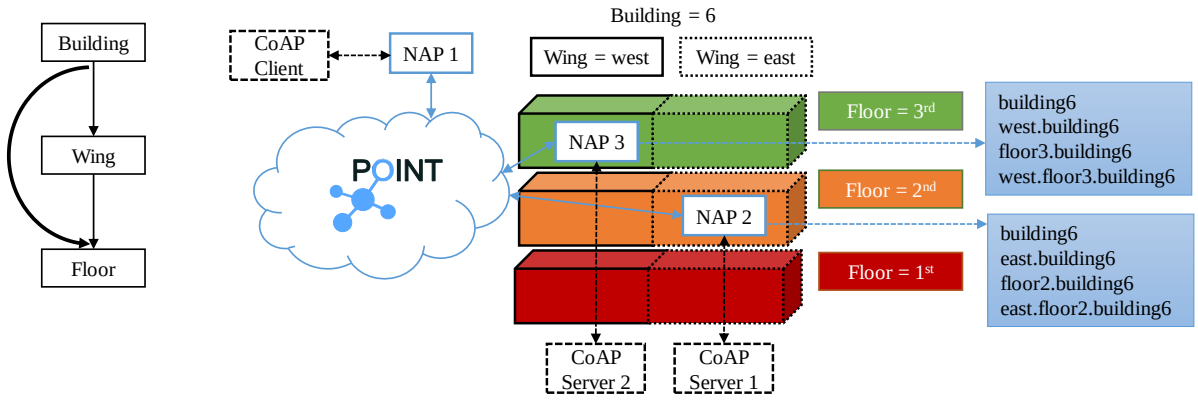



Figure 2: CoAP group name construction

To realize CoAP communication, the CoAP handler utilizes the namespace depicted in Figure 3. As shown on the left hand side of the figure, a request to a CoAP server is published with a random identifier under the scope, which is the hash of the server's Uri-Host (or a group name). Responses identifiers are hashes of the request packets.

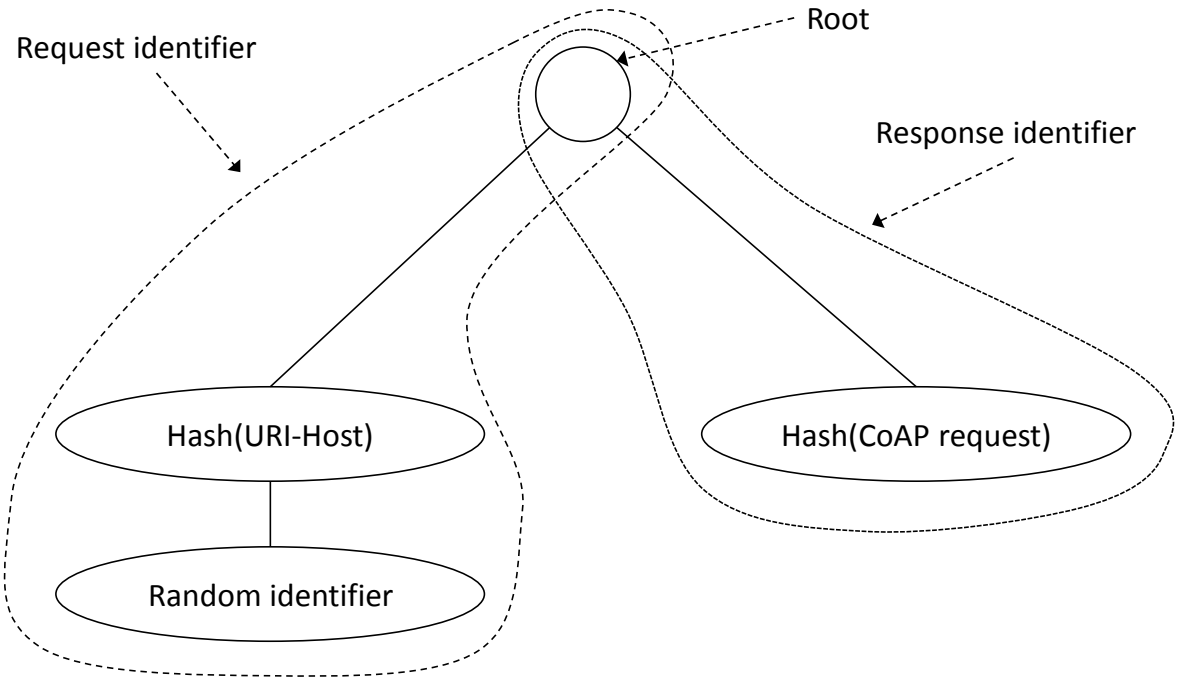



Figure 3: Namespace for CoAP-over-ICN

As explained above, CoAP runs on top of UDP, therefore the CoAP handler uses the proxy to terminate the UDP sessions between the CoAP client and client-side NAP (cNAP) and the CoAP server and server-side NAP (sNAP). This proxy decapsulates the CoAP payload from the UDP messages arriving from the CoAP client/server, and passes them to the CoAP handler for processing and transmission via the ICN network; conversely, it receives CoAP payloads from

the CoAP handler, which it encapsulates into UDP messages for transmission to the CoAP client/server.

## 2.1 Message Sequence Example

In Figure 4, we show an example message sequence chart (MSC) for a client request and a server response. The sNAP starts by subscribing to the server's hashed Uri-Host. On the low-level, the CoAP handler uses a scope prefix denoted by COAP_SCOPE_ID for the POINT communication, it creates separate scopes for each local CoAP server or CoAP group with the name: hash(Uri-Host) under COAP_SCOPE_ID and subscribes to those scopes. This means that sNAP will receive all information items published under those scopes. Currently, the list of all local CoAP servers along with their corresponding names is stored in a sNAP configuration file. In a real system, sNAP would query a CoAP resource directory (RD) for this information.

When the cNAP receives a CoAP request from the client (in this example: CoAP GET coap.aueb.gr), it advertises an information item with a random identifier under the scope: COAP_SCOPE_ID/hash(Uri-Host). This triggers the rendezvous process, the rendezvous system (RVZ) matches this publication with the server's subscription and asks the topology manager (TM) to calculate the appropriate forwarding identifiers, after which the RVZ forwards them back to the cNAP.

The cNAP stores the client token which is present in the CoAP request and strips token and CoAP message id from the request, in order to allow grouping of otherwise identical requests. Then the cNAP calculates the response identifier by hashing the CoAP request (which now doesn't contain the token and message id anymore). The cNAP also maintains a response identifier - list of client-side tokens mapping, such mapping is one-to-many since there might be multiple clients with different tokens that are using the same cNAP to observe the same CoAP resource. Finally the cNAP uses the publish_data_isub() primitive to publish the CoAP request under the scope: COAP_SCOPE_ID/hash(Uri-Host) and (implicitly) subscribe to receive the CoAP response using before mentioned response identifier.

In the next step the sNAP receives the CoAP request over the POINT network. The sNAP maintains a two way mapping between used server-side CoAP tokens and response identifiers (the sNAP generates a random 8-byte token before the first communication for each new response identifier). The sNAP will send the CoAP request over UDP to all CoAP servers that match the Uri-Host (which can also be a CoAP group name with multiple servers).

When the CoAP server responds, the sNAP determines the relevant response identifier based on the server's token. The sNAP strips the token and message id from the CoAP response, and publishes it under the response identifier.

Finally the response reaches the cNAP, which will forward it to all interested clients over UDP, these response packets will contain previously stored client-side token.

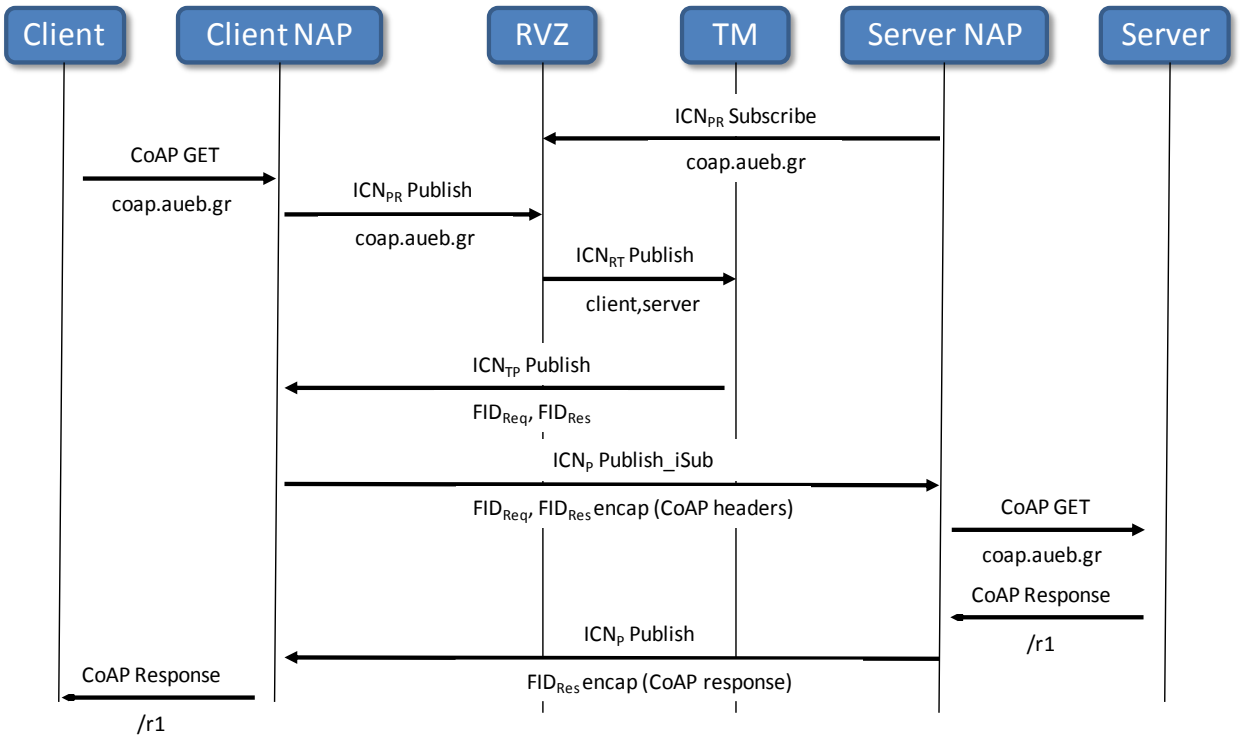



Figure 4: Message sequence chart for CoAP-over-ICN

CoAP group communication is realised in a following manner: each server NAP subscribes to all relevant group names (currently obtained through a configuration file). When there are multiple server NAPs subscribed to a (group) name, CoAP requests are forwarded to all of them, thus exploiting native multicast for CoAP group communication, without any centralized group configuration. sNAP also checks wherever there are multiple local servers corresponding to the same name, and forwards the requests to all of them.

In the reverse direction, a server NAP can use multicast to respond simultaneously to multiple clients asking for the same resource but attached to different NAPs. Unlike HTTP, this coincidental multicast behaviour is not an exceptional case in CoAP, it is rather the norm. This happens for two reasons. Firstly, a CoAP client may request a resource that is not yet available; in that case the CoAP server will send an ACK and respond later on, when the resource is ready. During this time, multiple requests for the same resource may arrive. Secondly, using the CoAP observe extension (RFC 7641) a CoAP client may indicate a permanent interest for a resource; in that case, every time a resource is updated, all interested clients will be informed.

In both cases, a server NAP forwards only the first request/interest registration to the CoAP server and all subsequent requests/interest registrations are suppressed (but stored in the NAP). Similarly, the server NAP suppresses all interest de-registrations, until the final interest is de-registered, when it forwards that de-registration to the CoAP server. As the CoAP specification recommends the use of a randomly generated token in order for a CoAP client to match (i) a delayed response to a request, (ii) a resource update message to an interest, client

NAPs store these tokens before forwarding such a request. When the corresponding response arrives, and before it is delivered to the CoAP client, the NAP restores the appropriate token.

Overall, the CoAP handler provides multiple benefits over the plain CoAP communication:

- Lower bandwidth overhead through multicast.
- Support for CoAP group communication without complicated DNS-based solutions and without need for IP multicast support by endpoints.
- Allowing multiple clients to observe the CoAP resource with a single observe relationship. This is important in real-world, since CoAP servers are usually resource constrained and do not support large number of observers.