H2020 iP Over IcN– the betTer IP (POINT)

Examples

Deploy a Resilient POINT Network

Author: Hasan M A Islam

1. Introduction

The main purpose of the CoAP Proxy is allow Constrained Application Protocol (CoAP) to run over the POINT platform. CoAP runs on top of the stateless UDP, therefore the CoAP proxy maintains states of the CoAP clients prior forwarding their requests to the POINT network. This allows the CoAP handler to forward CoAP responses back to the appropriate client.

In the current release, the CoAP proxy sends packets using the IP handler component of the NAP. In a future release, the CoAP proxy will be a part of the NAP's CoAP handler. In that case the proxy will map the CoAP message semantics to ICN messages.

2. Background

The CoAP proxy can act either as a Forward Proxy or a Reverse Proxy, both proxies can be implemented in a single binary. Further details are explained in RFC 7252: https://tools.ietf.org/html/rfc7252#section-5.7 .

1. Forward Proxy: the node performs requests on behalf of the client. The details are illustrated in https://tools.ietf.org/html/rfc7252#page-46
2. Reverse Proxy: the node behaves as if it was an original server. https://tools.ietf.org/html/rfc7252#page-47

The CoAP protocol contains four message types: Confirmable, Non-confirmable, ACK, and RST messages. More details can be found in the secion 4 of RFC 7252 (https://tools.ietf.org/html/rfc7252#section-4) while the CoAP message semantics are explained in section 3 of the RFC (https://tools.ietf.org/html/rfc7252#section-3).

3. The CoAP Forward/Reverse Proxy in the POINT platform

CoAP requests to a Forward proxy are made as normal Confirmable or Non-confirmable requests to the Forward proxy endpoint, but they specify the request URI in a different way. The request URI in a proxy request is specified as a string in the Proxy-Uri Option. Hence, a CoAP client interested in a resource "temperature", located in "www.aalto.fi" constructs a CoAP get request by setting the Proxy-URI option to "coap://www.aalto.fi/temperature" and sending it to the Forward proxy. Using a "coap-client" command tool provided by a libcoap library (https://libcoap.net/) such a request can be performed as follows:

```
$coap-client -P forwardproxy_address:port -v 100 -m get
coap://www.aalto.fi/temperature
```

In this request, the PROXY_URI option will be set in the coap packet and the option value will be "coap://www.aalto.fi/temperature". Read the Section 4 "Mapping CoAP messages to ICN messages" for more details.

The Forward Proxy parses the proxy URI and extracts the host (www.aalto.fi) and the path (temperature) parts of the resource. After parsing the proxy URI, the Forward Proxy creates a new request on behalf of the CoAP client using URI-HOST "www.aalto.fi" and URI-PATH "temperature" and forwards it to the CoAP Reverse Proxy. The CoAP Reverse Proxy forwards the request message to the original CoAP server. Finally, the CoAP server's response packet is sent back to the CoAP client through Reverse and Forward proxies. However, the role of the proxies in the current release can be defined as CoAP to CoAP proxy. The future release includes the role of CoAP to ICN proxy.

If the Forward Proxy is not set in the CoAP request, the request will be forwarded to the Reverse Proxy. Such a request can be performed as follows:

```
$coap-client  -v 100 -m get coap://www.aalto.fi/temperature
```

In this case, the Reverse proxy does not create any new packet as like the Forward Proxy.

3.1 Functional Description

Upon reception of the request message of the CoAP client, the following function defined in coap_proxy.h extracts the client information:

```
int on_client_msg(
                  struct client_node* node,
                  int client_sd,
                  char* hostname,
                  char *msg,
                  int len);
```

The node is the client_node data structure that stores the client information based on the token value:

```
struct client_node {
                  int sd;
                  struct client_node* next;
                  struct sockaddr_storage addr;
                  socklen_t addr_len;
                  unsigned char * token;
                  struct timeval last_seen;
```

```
            };
```

Upon reception of the request message from the CoAP client, the Forward Proxy invokes the `read_client` function to extract the client information and stores it to the list of client_node structure. This function is defined in the `clients.h` header file and the corresponding source file is implemented in clients.c source file.

```
int read_client(
                  int sd,
                  struct client_node** node,
                  char** msg, int* len);
```

If the CoAP client requests the message with the PROXY-URI option, the Forward Proxy extracts the Proxy-URI and creates a new coap request using the message translator of CoAP proxy. The `Proxy-URI` is splitted into `URI-Host`, `URI-Port`, `URI-Path` and inserted into the new packet. Then the proxy sends the request to the CoAP server as if it was the original client.

To send the request from Forward Proxy to Reverse Proxy, Forward Proxy invokes the following method which is defined in `coap_proxy.h`

```
int get_resource(
                  char* hostname,
                  char *msg,
                  int len);
```

The `hostname` is the address to which this node forwards the request.

When Reverse Proxy receives the request from Forward Proxy through the POINT platform, it forwards the request to the original CoAP server. The CoAP server sends the response to the Reverse Proxy which forwards the response back to the client over the reverse path using the following function:

```
void response_to_client(
                  unsigned char* token,
                   int tokenLen,
                   unsigned char* coapPacket,
                   int coapPacketLen);
```

The nodes in the reverse path look-up the appropriate client based on the token information resided in the CoAP response packet.

4. Mapping CoAP messages to ICN messages:

The CoAP proxy implements customized version of RFC 7252 to translate the CoAP messages to ICN messages. The implementation details and functional prototypes can be found in `coap.hpp` header file and the corresponding implementation can be found in `coap.cpp` source file.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Ver| T |  TKL  |      Code     |          Message ID           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Token (if any, TKL bytes) ...                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|   Options (if any) ...                                        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|1 1 1 1 1 1 1 1|    Payload (if any) ...                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 1: CoAP Message Format

4.1 Extracting fields from the CoAP Packet

`class CoAP_packet in coap.hpp provides all necessary functions to create a coap packet, to extract the CoAP protocol fields (Figure 1).`

`CoAP message types are defined by the following enum`

```
enum CoAP_msg_type{
                COAP_TYPE_CON = 0x00,
                COAP_TYPE_NON = 0x10,
                COAP_TYPE_ACK = 0x20,
                COAP_TYPE_RST = 0x30
            };
```

`The Request and Response Method Codes are defined by the following enum`

```
enum CoAP_method_code{
                COAP_METHOD_GET,
                COAP_METHOD_POST,
                COAP_METHOD_PUT,
                COAP_METHOD_DELETE,
            };
```

```
enum CoAP_response_code {
                  COAP_CODE_CREATED=0x41,
                  COAP_CODE_DELETED,
                  COAP_CODE_VALID,
                  COAP_CODE_CHANGED,
                  COAP_CODE_CONTENT,
                  COAP_CODE_BAD_REQUEST=0x80,
                  COAP_CODE_UNAUTHORIZED,
                  COAP_CODE_BAD_OPTION,
                  COAP_CODE_FORBIDDEN,
                  COAP_CODE_NOT_FOUND,
                  COAP_CODE_METHOD_NOT_ALLOWED,
                  COAP_CODE_NOT_ACCEPTABLE,
                  COAP_CODE_PRECONDITION_FAILED,
                  COAP_CODE_REQUEST_ENTITY_TOO_LARGE,
                  COAP_CODE_UNSUPPORTED_CONTENT_FORMAT,
                  COAP_CODE_INTERNAL_SERVER_ERROR=0xA0,
                  COAP_CODE_NOT_IMPLEMENTED,
                  COAP_CODE_BAD_GATEWAY,
                  COAP_CODE_SERVICE_UNAVAILABLE,
                  COAP_CODE_GATEWAY_TIMEOUT,
                  COAP_CODE_PROXYING_NOT_SUPPORTED,
                  COAP_CODE_UNDEFINED_CODE=0xFF
            };
```

`CoAP_packet(uint8_t *pkt, int pktlen)` returns a reference to `__coappkt` from the pkt byte array.

`uint8_t* getTokenValue()` returns the Token value from the CoAP packet which will be used to maintain the states of CoAP clients.

`CoAP_options* getOptions();` returns the Options from the CoAP packet. CoAP Options are stored in the following structure:

```
struct CoAP_options{
                  uint16_t delta;
                  uint16_t optionNumber;
                  uint16_t optionValueLength;
                  int optionLength;
                  uint8_t *optionValue;
}
```

To get the request URI form the the CoAP packet the following function is invoked.

```
int getResourceURIfromOptions(char** uri, int* urilen);
```

xFF in message format (Figure 1) indicates the start of the Payload.

4.2 Creating the CoAP Packet

Invoke the constructor: `CoAP_packet();`

Insert the Mandatory 4 byte Header:

```
int setVersion(uint8_t version);
void setType(CoAP_packet::CoAP_msg_type type);
int setTKL(uint8_t tkl);
int setTokenValue(uint8_t *token, uint8_t tokenLength);
int setMessageID(uint16_t messageID);
void setMethodCode(CoAP_method_code methodCode);
```

Next, insert options by invoking the following function:

```
int insertOption(uint16_t optionType, uint16_t optionLength, uint8_t*
optionValue);
```

N.B: the length of each Option follows the Delta Encoding (RFC 7252). Insert option one by one.

If you need to insert the Payload in the CoAP packet, insert xFF at the end of Options. It indicates the Payload Start marker.