Integration camp

mbed Cloud Edge architecture

Karthik Ranjan

Jarkko Jaakola

Janne Kiiskilä

2017-09-13

Taipei, Taiwan

Mbed Cloud Edge Product re-cap

Mbed Cloud Edge – Product Re-cap

Mbed Cloud Edge offers means to bridge devices to mbed Cloud.

For example

- Connecting existing legacy devices to mbed Cloud.
- Connecting non-IP devices to mbed Cloud (as mbed Cloud Client requires IP connectivity), for example Bluetooth LE.
- Connecting smaller-footprint devices to mbed Cloud that do not have enough capabilities to run the full mbed Cloud Client, for example some 6lowPan / Thread end-point.

Capability to do firmware update to mbed Cloud Edge itself.

Working together: Advantech and Arm work split

	Technology		Deliverables
ARM	• Security between gateway and Mbed Cloud (TLS, certificate) • Reporting of any errors in either the protocol translation interface to the legacy interface or connection to Mbed Cloud • Maintaining & reporting status of connectivity to Mbed Cloud	Covered in Workshop	• Mbed Cloud Client + example • Mbed Cloud Client documentation • Mbed Cloud Client Linux FW Update • Mbed Cloud Edge code • Mbed Cloud Edge Protocol Translator API
Advantech	• Onboard and pairing of LoRA devices to the gateway (protocol implementation)	✗	• Hardware and working SW for it. • Essentially the WISE-3610 SDK. • Protocol Translator implementation against Advantech's LoRa module
	• Protocol interface implementation between LoRA and Mbed Cloud Edge	✓	
	• Local control and management of LoRA	✗	
	• Offline data store & forward of LoRA endpoints	✗	
	• Monitoring status of protocol translation interface and connectivity to Mbed Cloud, including error reporting to the cloud • Failover handling to toggle (1) local control (2) store & forward	✓	
	• Factory provision keys into the gateway	✓	
	• Core gateway security	✗	

2nd release available – 0.1.1

First releases have now been made available.

It’s by no means perfect or ready to production quality yet.

This is still work in-progress, but – it should nevertheless enable the PT development work for connected devices.

CHANGELOG in the repository has lots of details.

You can:
- Get mbed Cloud Edge itself to the mbed Cloud
- You can register PT
- You can add devices to PT and mbed Cloud
- You can GET and PUT values to devices.

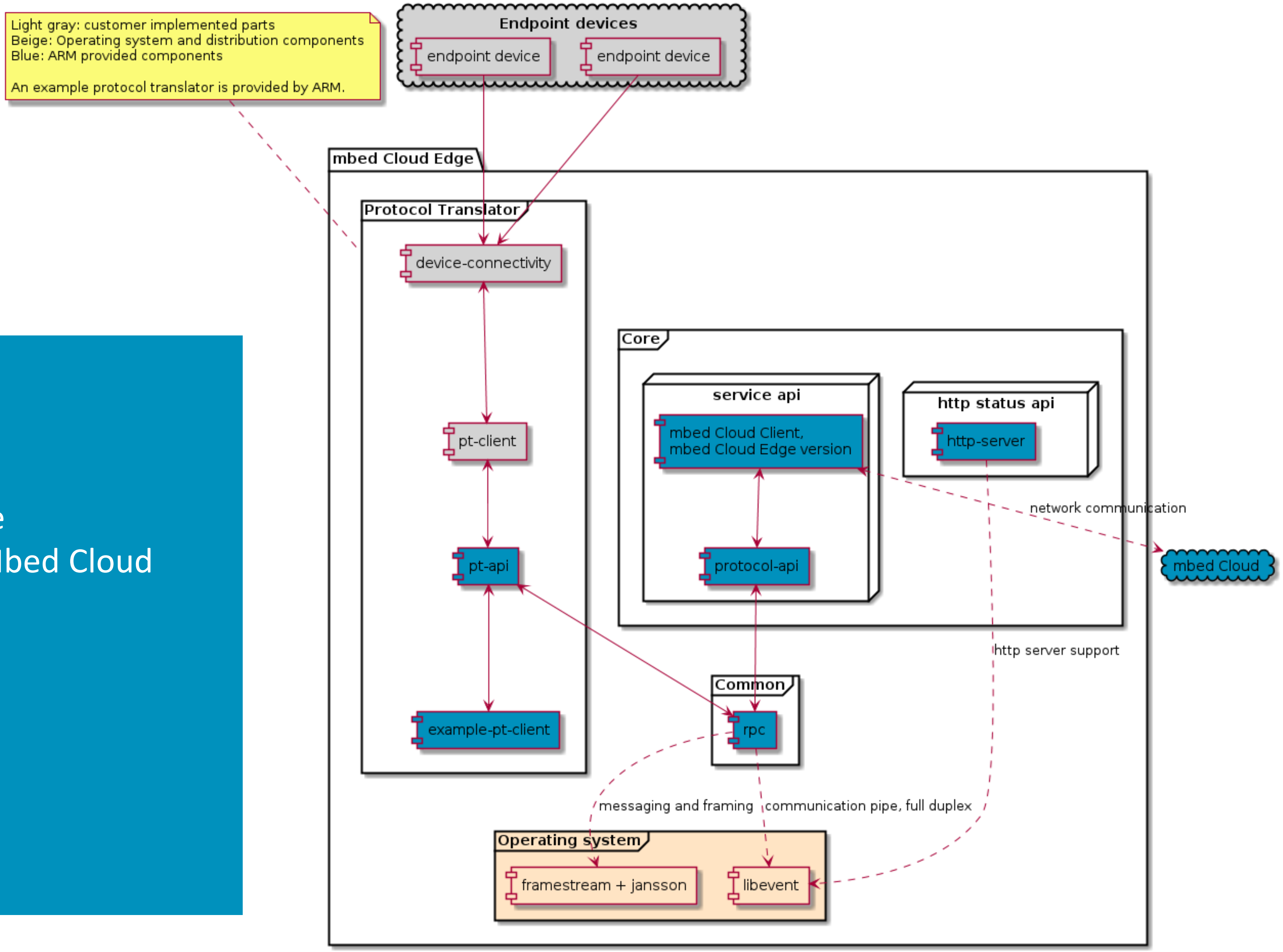
Known issues:
- POST/DELETE missing.
- Observe device resources yet, but you can observe resources for that device via the mbed Cloud Edge device itself.
- Endpoint_type can’t be used via REST API yet.
- There are issues with high volume device creation.
- Configuration is not simple or centralized yet.
- Moving end points not supported yet.

mbed Cloud Edge
High level architecture

Architecture overview picture

5 main modules

1. Mbed Cloud
2. Mbed Cloud Edge Core
   - Which includes Mbed Cloud Client
3. Protocol API
4. Protocol Translator
5. End points/devices.



Architecture overview

The full system of mbed Cloud Edge contains five main components

- **Mbed Cloud**
  - As current mbed Cloud but has extended support to handle and maintain mbed Cloud Edge mediated devices.
  - The 1st required changes are already in mbed Cloud production.
- **Mbed Cloud Edge Core**
  - Implements the communication between protocol translator and mbed Cloud.
  - Implementation is based on mbed Cloud Client, it is an extended version of it.
    - This mbed Cloud Client has the **full Cloud Client capabilities**.
    - Firmware update of the mbed Cloud Edge product itself is done via the normal mbed Cloud Client Firmware update.
    - In similar manner also the production certificate injection is done via the Factory provisioning mbed Cloud Client uses – there is nothing mbed Cloud Edge specific about those aspects.

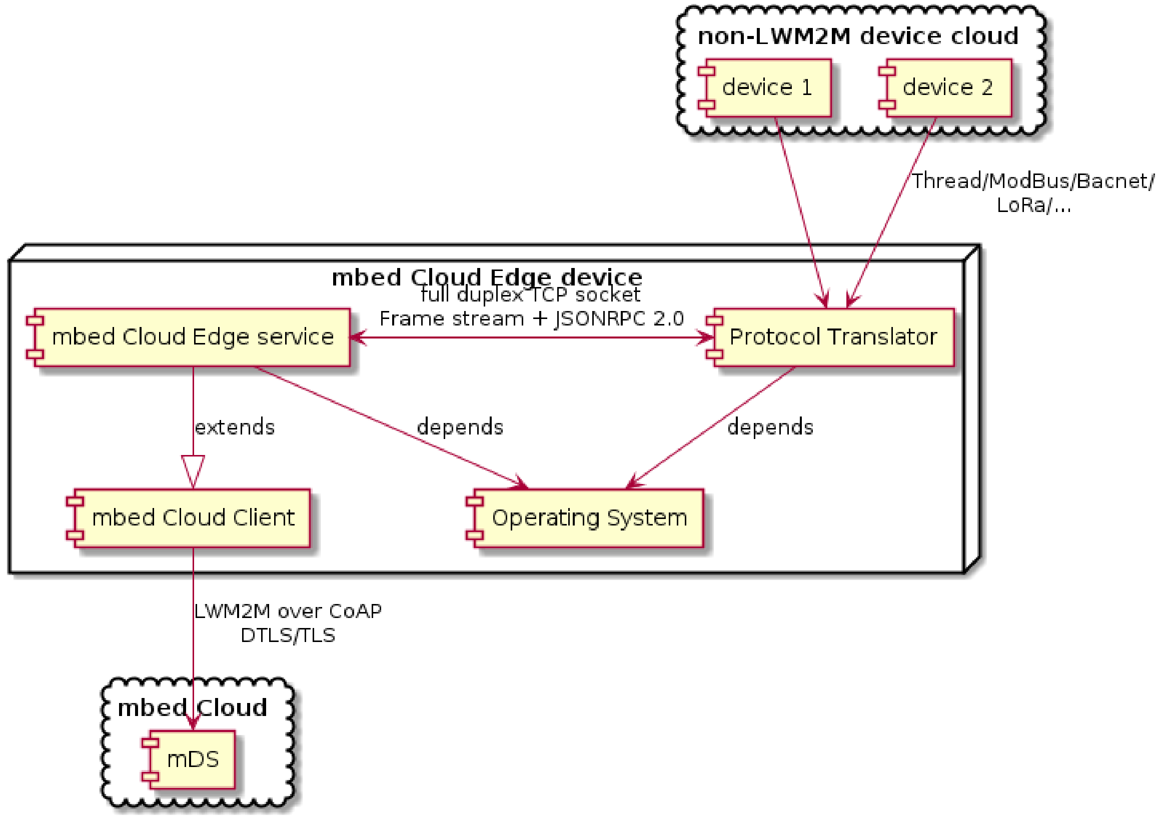
Architecture overview continued

- **Protocol translator Application Programming Interface (API)**
  - Mbed Cloud Edge provides protocol translator API and an example client.
  - Does not provide tools or means to handle connectivity to endpoint devices.
    - Those are always customer connectivity specific.
  - Provides API to device resources as IPSO Smart Objects data structures.
- **Protocol translator implementation**
  - **Customers need to implement their own endpoint device handling, i.e. connectivity to devices.**
  - PT must also map device resources into IPSO Smart Objects,
    for example thermostats have a pre-defined IPSO Smart Object `3300`.
- **Non-LWM2M endpoint device(s)**
  - These are external connected devices, e.g. LoRa, BLE, BACNet, narrow-band IOT, etc.
  - Customer is responsible of the connectivity to mbed Cloud Edge device.

Architecture overview

Components

- mbed Cloud Edge service and protocol translator communicates with full duplex local socket connection.
- Operating system must provide some hard dependencies* for mbed Cloud Edge.



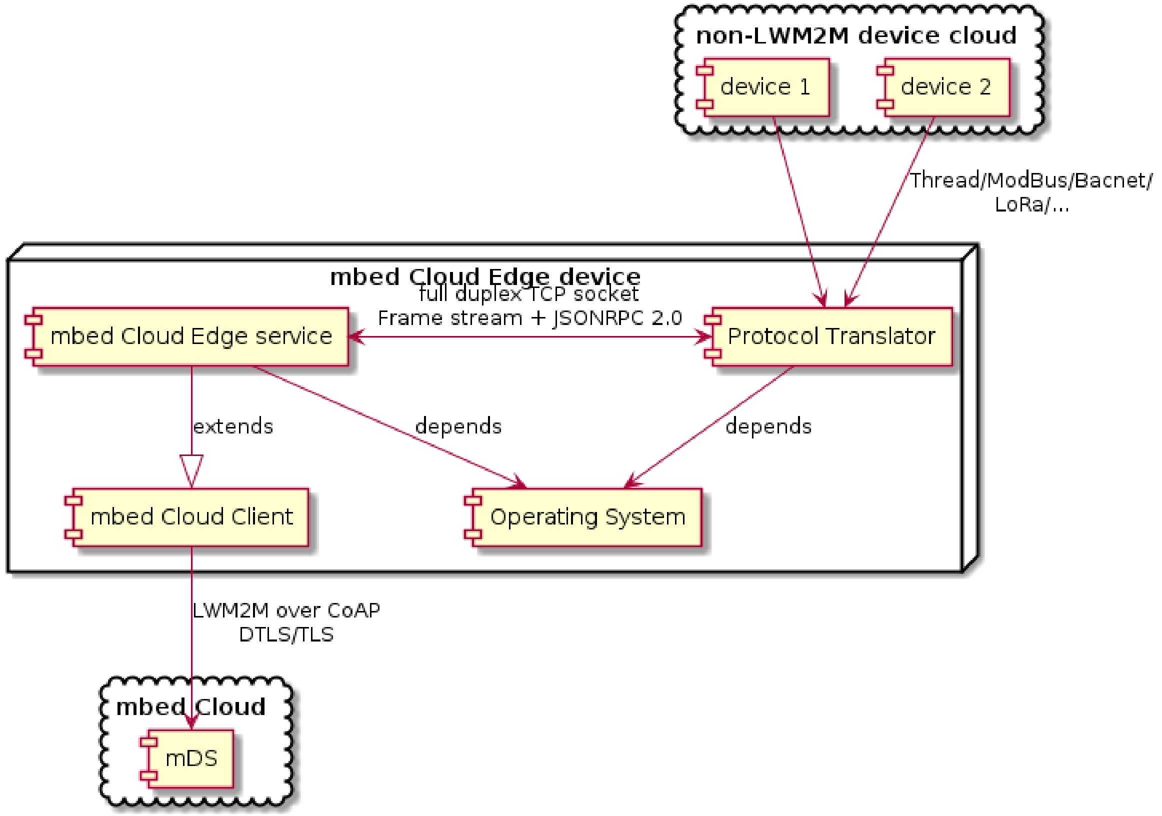
* Refer to release notes & slide internal components.

arm

Architecture overview

Components

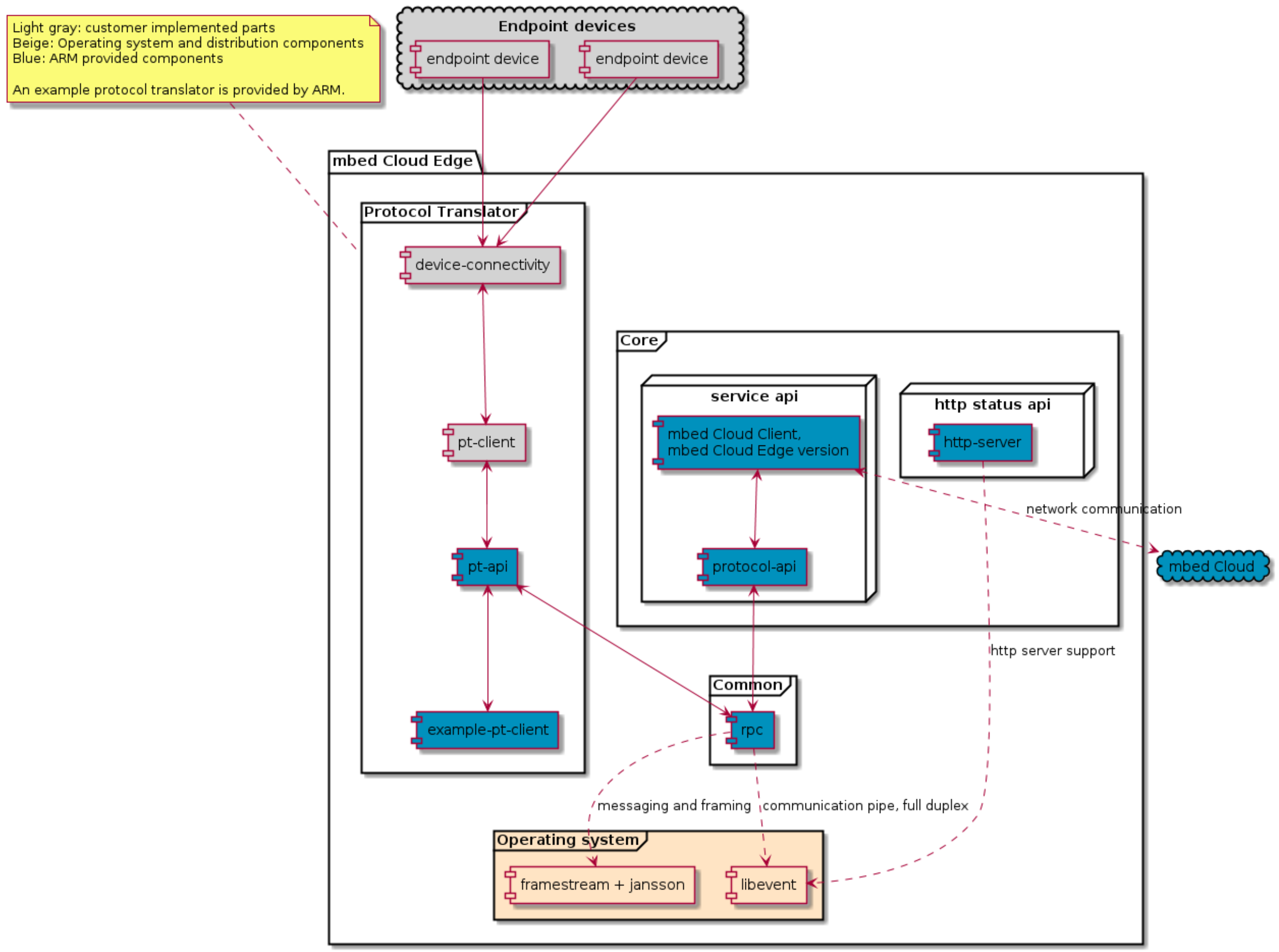
- Mbed Cloud Client to Mbed Cloud uses COAP over TLS/DTLS.
    - Registration message link format has Edge proprietary extension!
- ARM does not handle the connectivity for the devices – PT must implement that.



Internal components

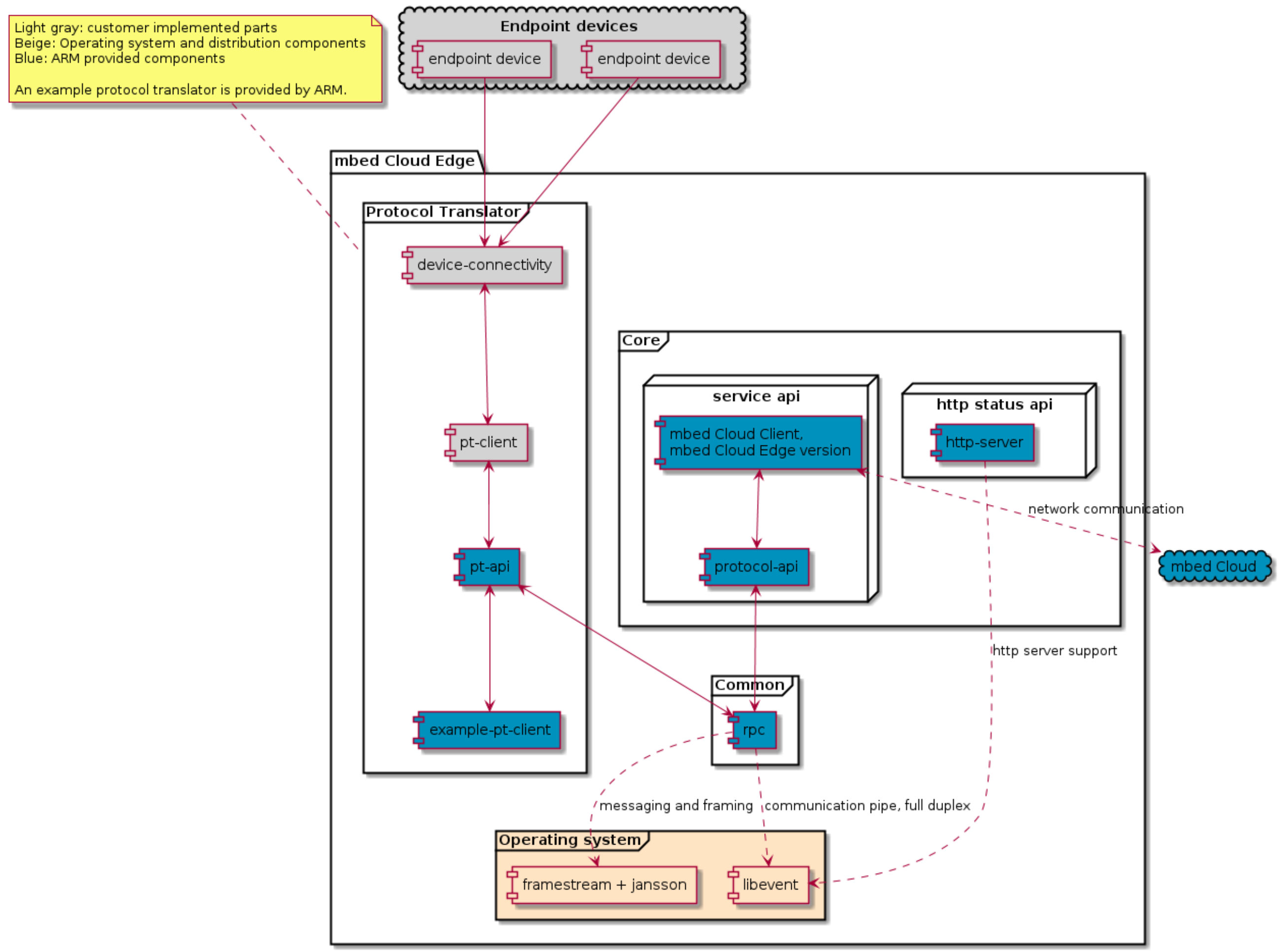
Protocol translator API

- pt-api (client side)
  - Provides a set of functions to initiate actions on mbed Cloud Edge. Callbacks are used to communicate back to customer code.
  - Provides RPC interface to protocol translator that mbed Cloud Edge service can call.
  - **Eventloop! Callback code must not block processing.**
- protocol-api (Edge Core)
  - Runs in libevents event loop.
  - Provides RPC interface to mbed Cloud Edge that protocol translator can call.



Internal components

- Dependencies
  - libevent 2.x for socket communication
  - frstrm for framing messages
  - jansson
  - jsonrpc (included as source) for message format
- Common
  - Eventloop based PRC framework, tracks sent messages and corresponding responses.

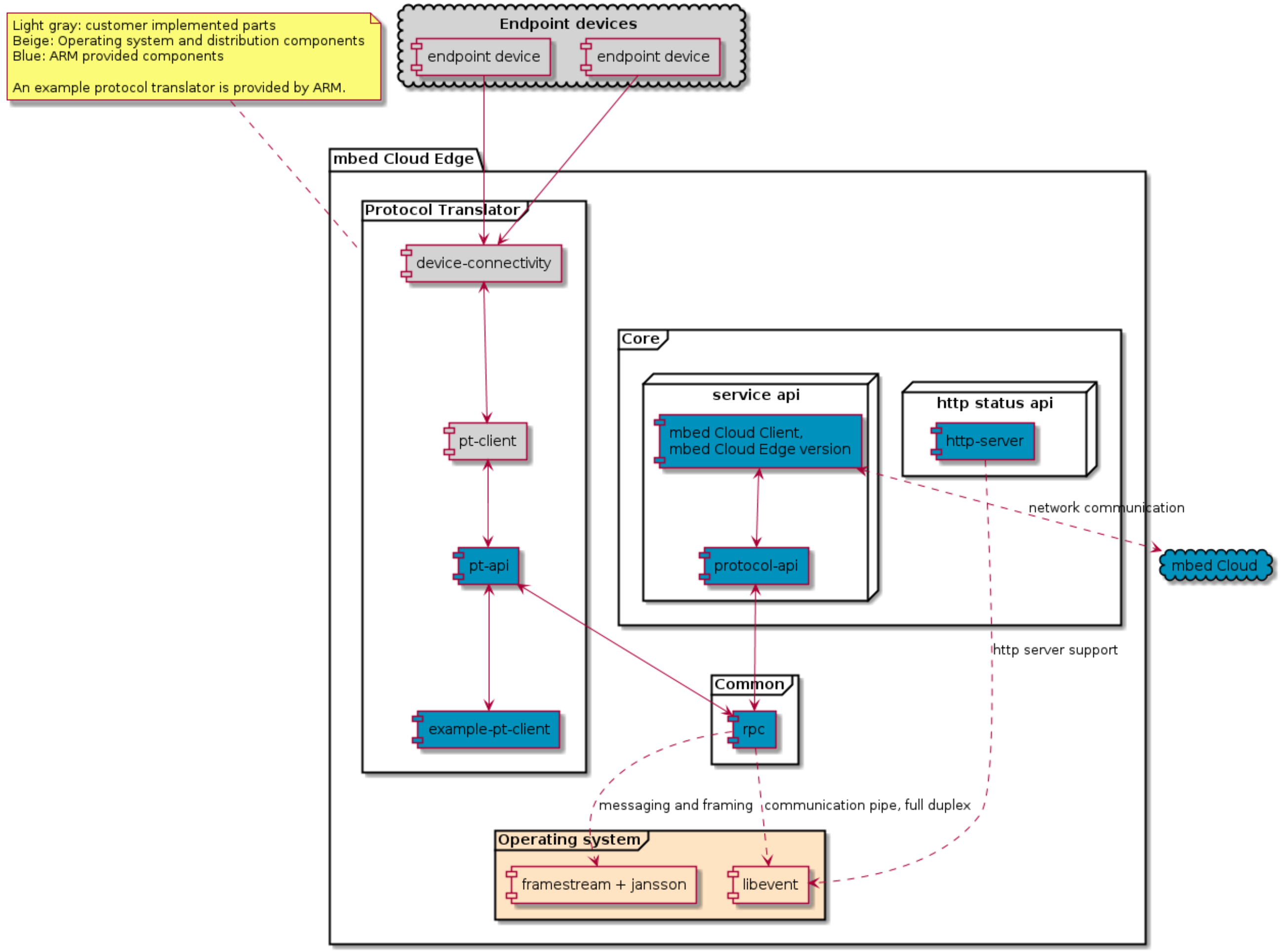


Light gray: customer implemented parts
Beige: Operating system and distribution components
Blue: ARM provided components

An example protocol translator is provided by ARM.

Internal components

- mbed Cloud Client - Edge version
  - Implemented in mbed Cloud Client and mbed Cloud Edge projects.
  - Changes also in mbed Cloud Client to handle the mbed Cloud Edge special cases.
- pt-client and device connectivity
  - Customer implemented part of protocol translator.
  - Implements the callbacks for protocol translator and mbed Cloud Edge services protocol API.
  - The customer callbacks will offload the processing and control to implementation responsible of communicating the state to endpoint devices.



mbed Cloud Edge Core

mbed Cloud Edge Core

Provides an API for protocol translators to call. Provided JSON-RPC API is not directly visible from protocol translators (hidden behind C API).

Similar kind of messaging queue and message handling is utilized in mbed Cloud Edge Core as in the protocol translator API.

The internal JSON message is parsed and handled in the mbed Cloud Edge Core. The resulting translation to mbed Cloud Client datastructures is implemented in the Core.

Mbed Cloud Client is an important part of the mbed Cloud Edge service as it handles all the connectivity and data passing to mbed Cloud.

NOTE: mbed Cloud Edge Core code is something that should just work out of the box apart from configuration by customers (certificates, credentials, Cloud Edge resources, etc).

mbed Cloud Edge

Operations that mbed Cloud Edge service MUST support

- From mbed Cloud REST API it must be possible to GET the values from endpoint devices.
  - Protocol translators must write values to mbed Cloud Edge service. Core will store values in mbed Cloud Client structures.
- From mbed Cloud REST API it must be possible to observe the values (GET) from endpoint devices.
  - All protocol translated endpoint resources are observable. The mbed Cloud Client and mDS handle the observations.
    - NOTE! Right now in 1st version the observations for devices do not work yet, they only work on the resources in the actual mbed Cloud Edge itself. Will be fixed soonest.
- From mbed Cloud REST API it must be possible to PUT/POST/DELETE values to endpoint devices.
  - PUT has been implemented. More complicated than GET, as there are asynchronicity issues with sleepy end-points. Details later on in flow charts.
  - POST/DELETE still under work, but will follow the same schema as PUT.
  - Some cases may be just pass through, e.g. Object delete.

mbed Cloud Client – Edge version

mbed Cloud Client - Edge version 1/2

Mbed Cloud Edge is written in C and there is a client wrapper to bridge the C++ code to C.

Also there are number of changes to mbed Cloud Client to have the needed functionality for mbed Cloud Edge:

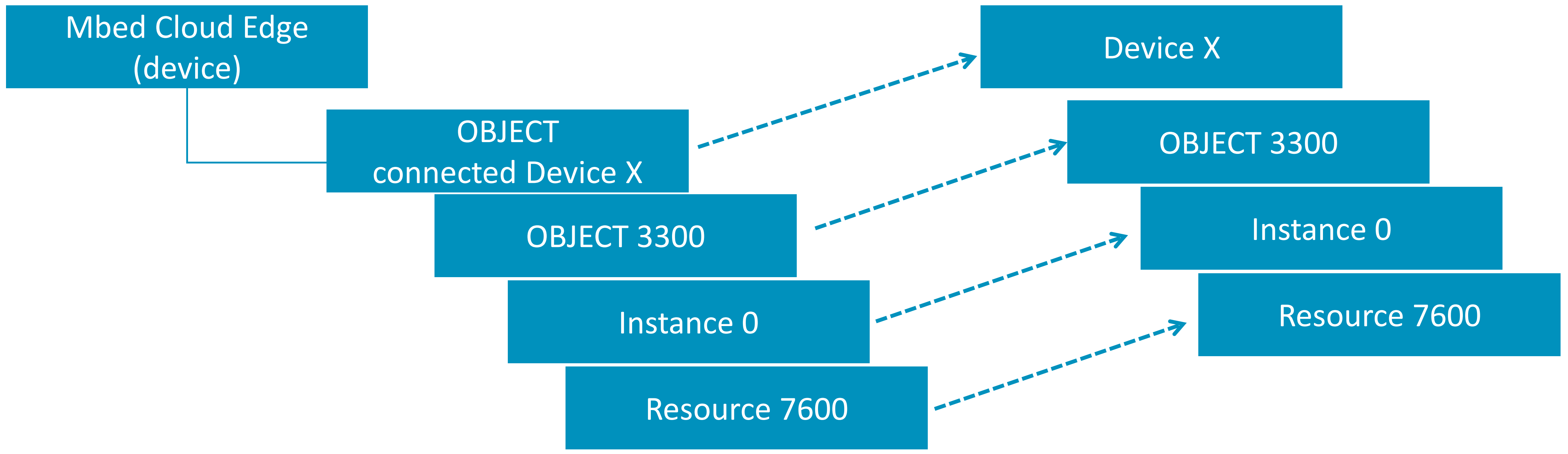
- Re-registration support, to have the translated devices registered to mbed Cloud the full registration message must be sent. Registration update support is on the roadmap.
- Link format changes in the registration message to explicitly identify mbed Cloud Edge resources as endpoint devices in mbed Cloud.
- Resource removal support implemented to mbed Cloud Client, mbed-client does have it but the mbed Cloud Client wrapper did not yet have it.

Mbed Cloud Edge Core implementation has mbed Cloud Client running and the mbed Cloud Edge device itself is also as a normal LWM2M device.

mbed Cloud Client – Edge version 2/2

Connected device is presented as an object in mbed Cloud Edge devices resource structure.

The link format change and mbed Cloud changes were implemented to represent these resources as individual endpoint devices in the mbed Cloud.



Protocol Translator Use-case-flows

We supply one example – pt-example

That show cases the available functionality

Please look up the file `pt-example/client_example.c`.

It contains a good starting point for any protocol translator.

It has comments ”customer code” – these are the hook points.

It’s main() has this:

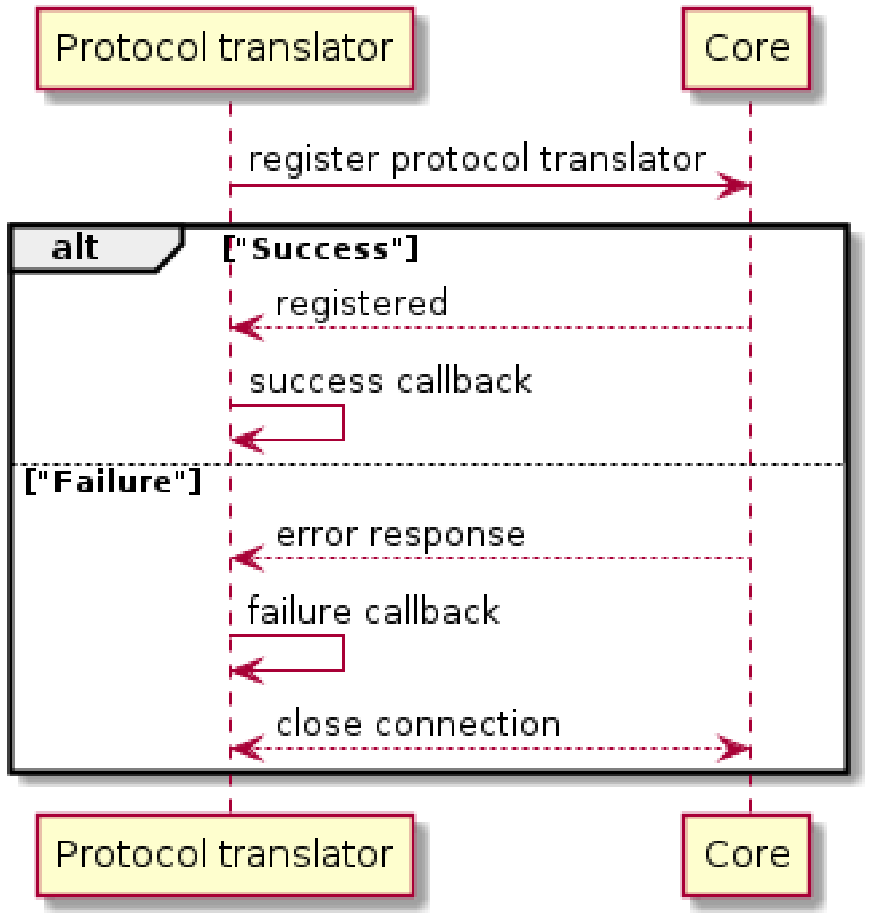
```
pt_client_start(port, (char*)pt_name,
    connection_ready_handler, received_write_handler);
```

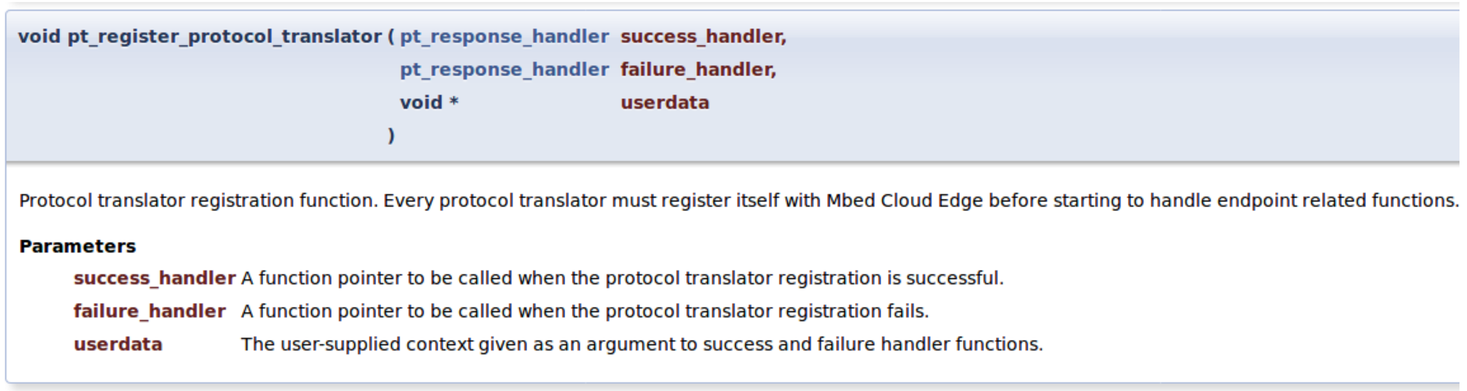
This is where the PT things start.

1) `connection_ready_handler` will register the PT.

After that – everything is event based.

Register PT





```
void pt_register_protocol_translator ( pt_response_handler  success_handler,
                                       pt_response_handler  failure_handler,
                                       void *               userdata
                                     )
```

Protocol translator registration function. Every protocol translator must register itself with Mbed Cloud Edge before starting to handle endpoint related functions.

Parameters

success_handler	A function pointer to be called when the protocol translator registration is successful.
failure_handler	A function pointer to be called when the protocol translator registration fails.
userdata	The user-supplied context given as an argument to success and failure handler functions.

Our pt-example has this code.

```
void connection_ready_handler()
{
    /* Initiate protocol translator registration */
    pt_register_protocol_translator(
        protocol_translator_registration_success,
        protocol_translator_registration_failure,
        NULL);
}
```

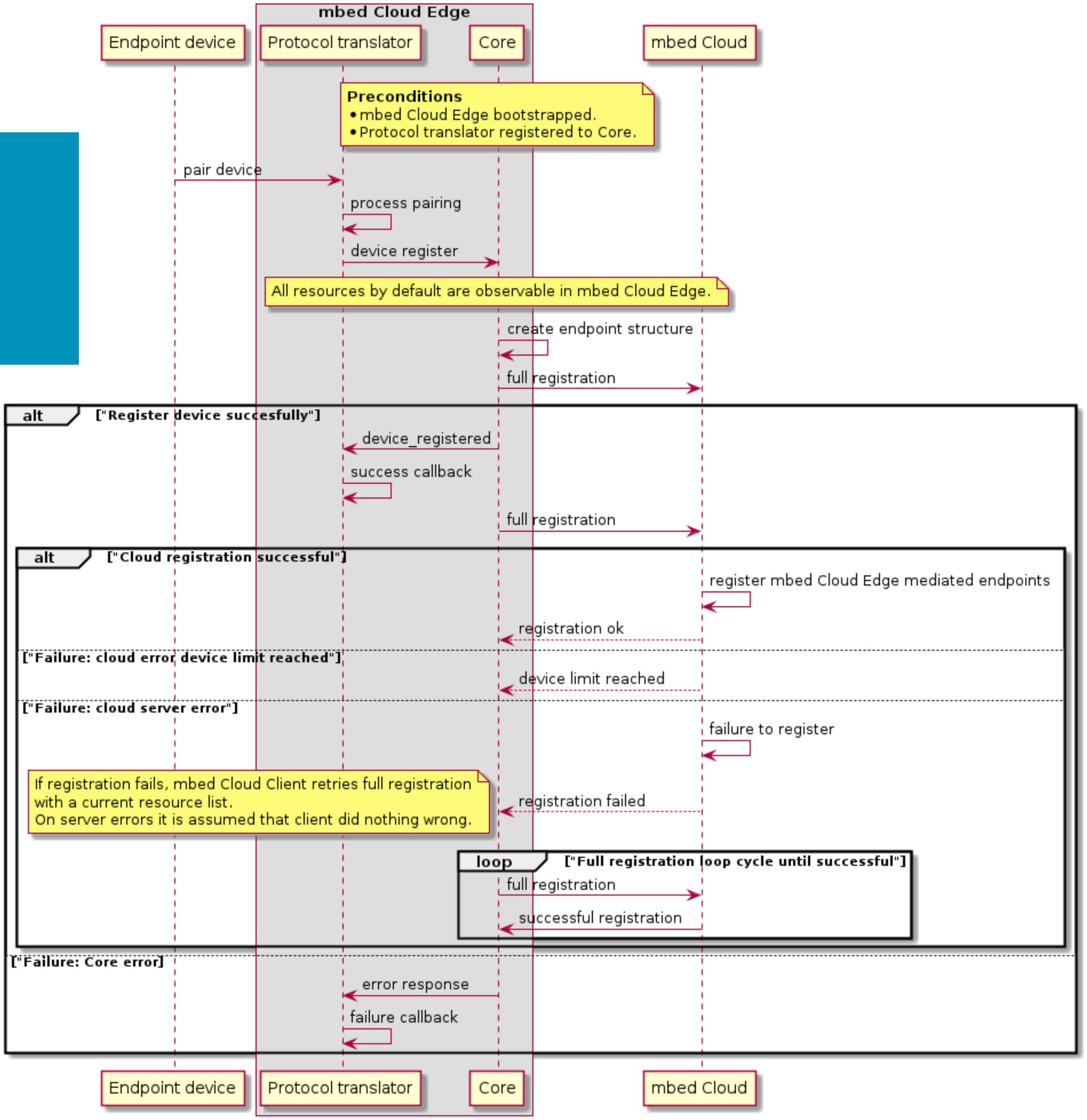
Add end-points

The beginning part:
– how to find the devices
– potentially pair/authenticate them
PT needs to handle all that.

End-point structure describes the objects, object instances & resources the device has.

Call-backs are used to handle failure and success.

Consider the callbacks interrupt context!
- If your actions take a long time, kick the job to a worker thread.



mbed Cloud Edge
Endpoint device
Protocol translator
Core
mbed Cloud
Preconditions
• mbed Cloud Edge bootstrapped.
• Protocol translator registered to Core.
pair device
process pairing
device register
All resources by default are observable in mbed Cloud Edge.
create endpoint structure
full registration
alt
["Register device succesfully"]
device_registered
success callback
full registration
alt
["Cloud registration successful"]
register mbed Cloud Edge mediated endpoints
registration ok
["Failure: cloud error device limit reached"]
device limit reached
["Failure: cloud server error"]
failure to register
If registration fails, mbed Cloud Client retries full registration with a current resource list. On server errors it is assumed that client did nothing wrong.
registration failed
loop
["Full registration loop cycle until successful"]
full registration
successful registration
["Failure: Core error]
error response
failure callback
Endpoint device
Protocol translator
Core
mbed Cloud

Code from pt-example:

1st, create device PT structure

```
/**
 * \brief Creates a pt_device structure.
 *
 * @device_is The unique ID of the device.
 * @lifetime The device lifetime in seconds.
 * @queuemode The queue mode.
 *
 */
struct pt_device* pt_create_device(char* device_id, uint32_t lifetime, char* queuemode)
{
   struct pt_device *device = (struct pt_device *)malloc(sizeof(struct pt_device));
   device->device_id = device_id;
   device->lifetime = lifetime;
   device->queuemode = queuemode;
   device->objects = pt_object_list_create();

   return device;
}
```

NOTE! Device ID must be:
- Unique (to all of your devices) AND
- Persistent (must remain the same, always)

Lifetime and queue mode are currently <u>not</u> supported.

Code from pt-example:

2ndly, we create the objects/object instance/resources.

```
/**
 * \brief Creates objects representing example M2M LWM object structure.
 * \param temperature is value for creating example resource that is stored as opaque value at
 *       path `/<example-device-id>/5432/0`
 *       The opaque value will become e.g. "100 F" when given 100 as input temperature.
 */
pt_object_list_t *create_device_objects(int temperature) {
   const char *opaque_data_string ="Temp          ";

   /* IDs */
   uint16_t object_id = 5432;
   uint16_t opaque_id = 0;
   uint16_t object_instance_id = 0;

   pt_object_list_t *objects = NULL;
   pt_object_t *object = NULL;
   pt_object_instance_t *instance = NULL;
   pt_resource_opaque_t *resource = NULL;
   char *data = alloc_string_from(opaque_data_string);
   sprintf(data + 5, "%d F", temperature);
   unsigned int operations = 0;
   operations |= (OPERATION_READ | OPERATION_WRITE | OPERATION_EXECUTE | OPERATION_DELETE);

   object = pt_object_create(object_id);
   objects = pt_object_list_create();
   instance = pt_object_instance_create(object_instance_id);
   pt_object_list_add_object(objects, object);
   pt_object_add_instance(object, instance);

   resource = pt_resource_create_opaque(opaque_id, operations, data, strlen(data) + 1);
   pt_object_instance_add_resource(instance, (pt_resource_t *)resource);
   return objects;
}
```

5432/0/0 will be created.

Allowed operations for the object. Could be a subset, for example READ only.

Code from pt-example:

3rd, we register the device – using the device PT info and object list as input.

```
/**
 * \brief Register the device to Mbed Cloud Edge.
 *
 * The callbacks are run on the same thread as the event loop of the protocol translator client.
 * If the related functionality of the callback does some long processing the processing
 * must be moved to worker thread. If the processing is run directly in the callback it
 * will block the event loop and therefore it will block the whole protocol translator.
 *
 * \param device_id_string The unique identification of the device.
 * \param success_handler The function to be called when the device registration succeeds
 * \param failure_handler The function to be called when the device registration fails
 * \param handler_param The parameter that is passed to the success or failure handler.
 * \param temperature The initial temperature of the device
 */
static void register_device(
    const char *device_id_string,
    pt_response_handler success_handler,
    pt_response_handler failure_handler,
    void *handler_param,
    int temperature)
{
   struct pt_device *device = create_device(device_id_string);
   pt_object_list_t *objects = create_device_objects(temperature);
   pt_device_add_object_list(device, objects);
   pt_register_device(device, success_handler, failure_handler, handler_param);
   pt_object_list_destroy(objects);
   pt_device_free(device);
}
```

Create device, it’s objects, add them to the list and ask to register (with callbacks).

Once the objects are sent, they are not needed anymore (by default) – if it succeeds, they will be the mbed Cloud Client object list.

GET – write to Cloud

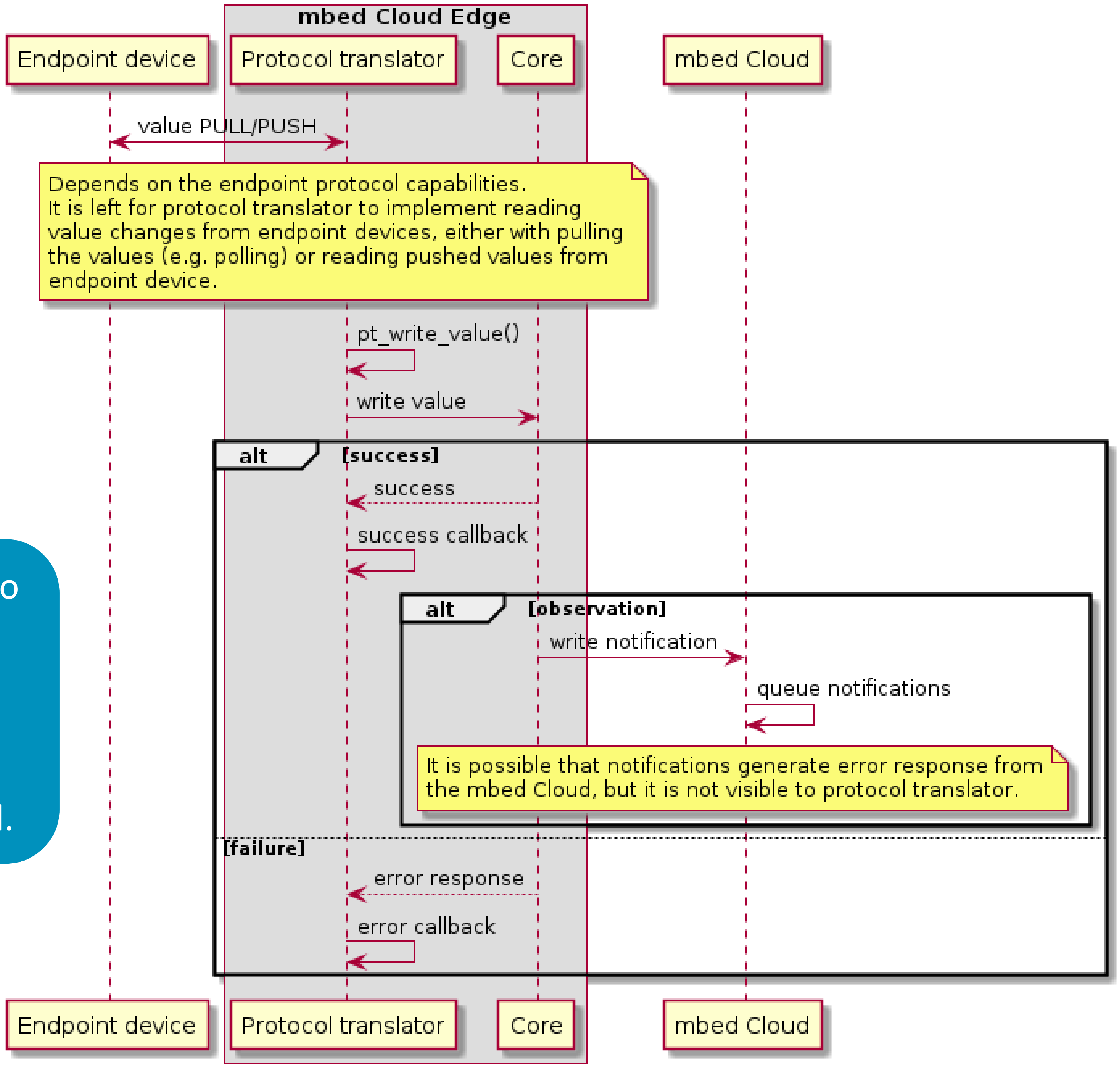
Flow chart

It fully depends on the connected device if you either

a) Get some interrupt or similar to inform of a new resource value OR
b) You have to poll the devices periodically.

Nevertheless, once you have the value in the PT to write, call `pt_write_value()`.

- There can be failures (connection broken, ...).
- Callbacks are used again.
    - Again, interrupt context!
    - Don't take too long OR use worker thread.



Depends on the endpoint protocol capabilities.
It is left for protocol translator to implement reading value changes from endpoint devices, either with pulling the values (e.g. polling) or reading pushed values from endpoint device.

It is possible that notifications generate error response from the mbed Cloud, but it is not visible to protocol translator.

GET – example code

Pre-emptively writing the data to the mbed Clout

```
/**
 * \brief Writes a value to device with given device id.
 *
 * The callbacks are run on the same thread as the event loop of the protocol translator client.
 * If the related functionality of the callback does some long processing the processing
 * must be moved to worker thread. If the processing is run directly in the callback it
 * will block the event loop and therefore it will block the whole protocol translator.
 *
 * \param device_id_string The unique identification of the device.
 * \param temperature The temperature of the device.
 */
void write_value(const char *device_id_string, int temperature)
{
   struct pt_device *device = create_device(device_id_string);
   pt_object_list_t *objects = create_device_objects(temperature);
   tr_info("write_value temperature=%d", temperature);
   pt_device_add_object_list(device, objects);
   pt_write_value(device, device->objects, write_value_success, write_value_failure, NULL);
   pt_object_list_destroy(objects);
   pt_device_free(device);
}
```

Create target device, objects, object info, resource we want to update.

Call pt_write with the device, device object and the callbacks for success and failure.
You can also give some callback parameter, if need be.

Free up the device, object etc. after doing the write.

PUT

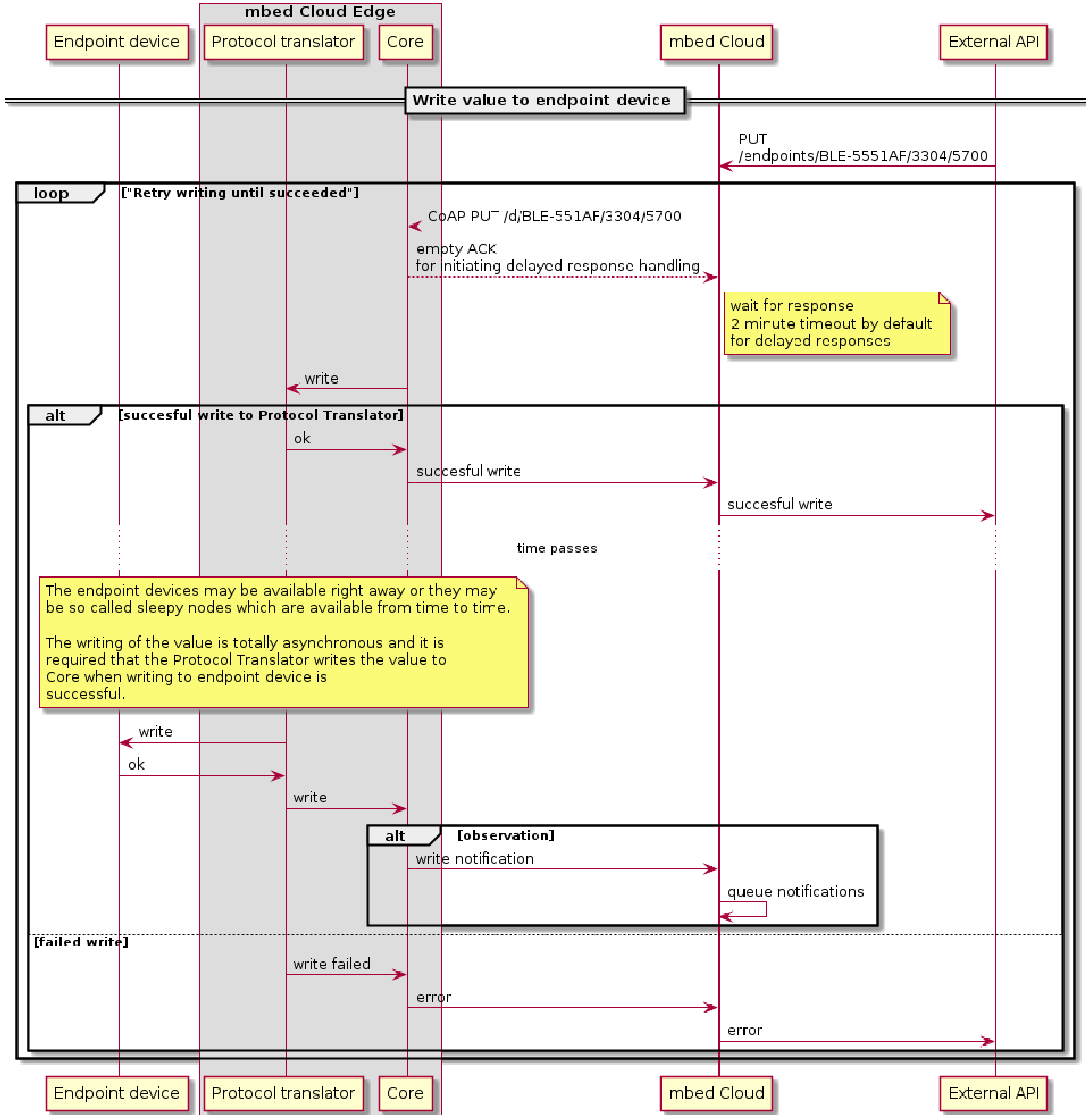
Cloud writes to device

PUT is more trickier if you do not have constant connection to the device.
Cloud will wait maximum 2 minutes for the ACK for delayed response.

A PUT will trigger write callback in the PT.
The value in the MCE Core WILL NOT CHANGE.

If the end point is not reachable, you must queue the request to the PT. PT needs to do the write once it has connection to the device again.

You must do a `pt_write_value` (i.e. PUT) to the device in order to change that value in the MCE and reflect it back as a confirmation to the mbed Cloud.



wait for response
2 minute timeout by default
for delayed responses

The endpoint devices may be available right away or they may be so called sleepy nodes which are available from time to time.

The writing of the value is totally asynchronous and it is required that the Protocol Translator writes the value to Core when writing to endpoint device is successful.

PUT – write data to device – example code

Remember the main() function?

pt_client_start(port, **(char*)pt_name, connection_ready_handler, received_write_handler);**

```
/**
 * \brief Implementation of handler for write messages received from mbed Cloud Edge Core.
 *
 * \param device* Received data from write message. The ownership of the data structure is passed
 * to this callback.
 */
void received_write_handler(struct pt_device *device)
{
   tr_info("mbed Cloud Edge write to protocol translator.");
   tr_info("Write back to mbed Cloud Edge the incoming data.");
   pt_write_value(device, device->objects, write_value_success, write_value_failure, NULL);
   /*
    * Free the structures, the ownership is transferred to this callback.
    */
   pt_object_list_destroy(device->objects);
   free(device->device_id);
   pt_device_free(device);
}
```

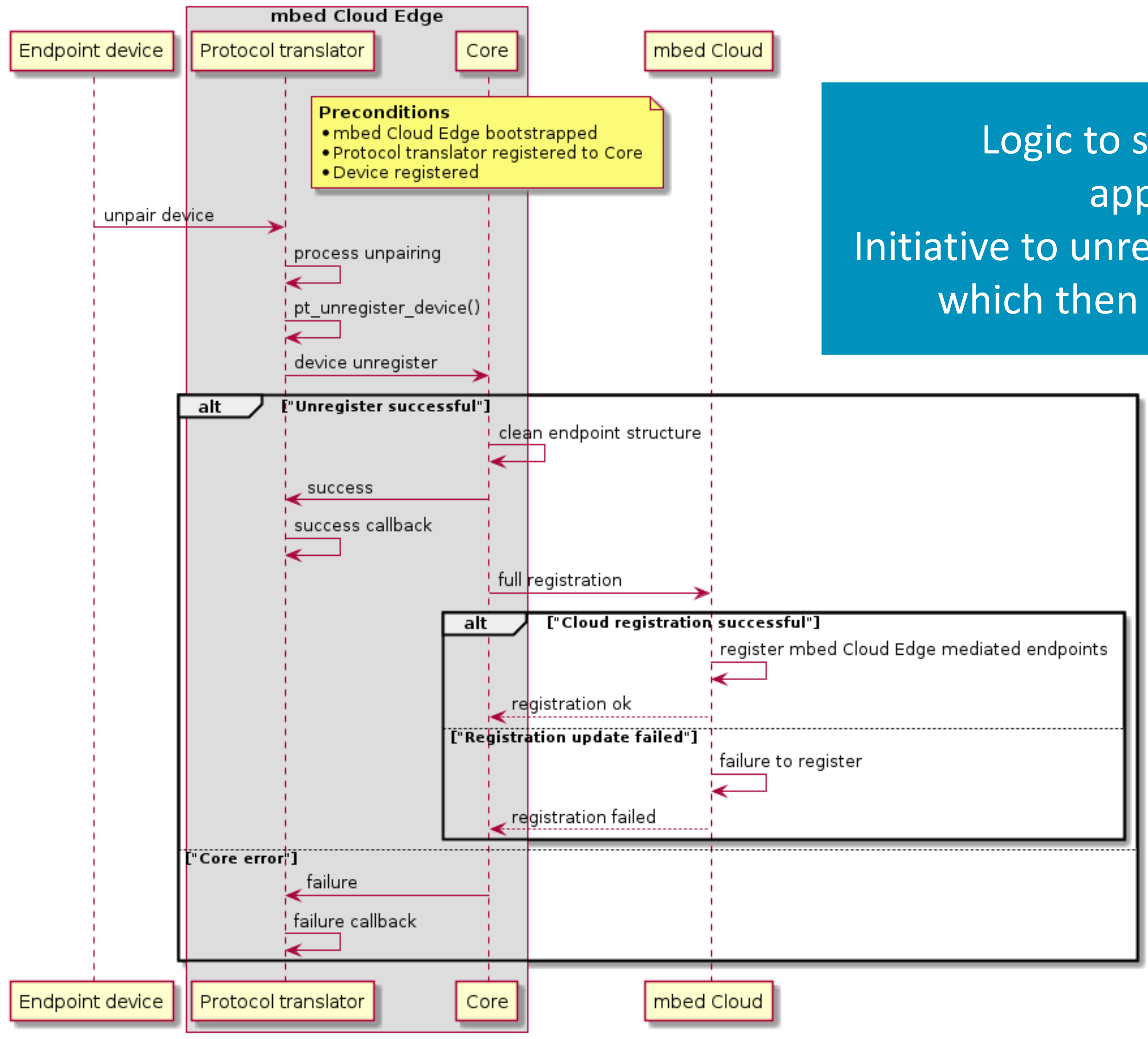
This callback gets activated.

The device information has links to the object , object instance, resource & value.

PT needs to map those to the real device resources and do the writing there.

If immediate write is not possible – quite a lot of logic needs to be implemented for the queueing.

Remove (un-register) end-point



Logic to spot the device needs removal is application/protocol specific.
Initiative to unregister has to come somehow to the PT, which then initiates the unregister procedure.

Unregister device – code sample

```
/**
 * \brief Unregisters the test device
 */
void unregister_test_device()
{
    struct pt_device *device = create_device(client_config_get_example_device_name());
    pt_unregister_device(device, device_unregistration_success, device_unregistration_failure, /* userdata */ NULL);
    pt_device_free(device);
}
```

Fairly straight-forward – create device structure for PT and ask for unregister.

POST, DELETE

Not supported yet.

POST and DELETE have not been implemented yet.

They will come in upcoming releases.

However, the logic with these will be very much similar to what the PUT operation is having.

- Callback to be registred for the operation.
- Callback gets the device pointer.

Special considerations

Sleepy or moving things are not that easy

If you have to queue PUTs, POSTs etc. to non-connected/sleepy devices, you must ensure you will not run out of memory with the queue.

- Once devices move or run out of battery – those events need to be flushed out from the queue.
- Device lifetime (re-registration period) and a timestamp for events,
- Periodic cleanup if (timestamp+lifetime) > time now.

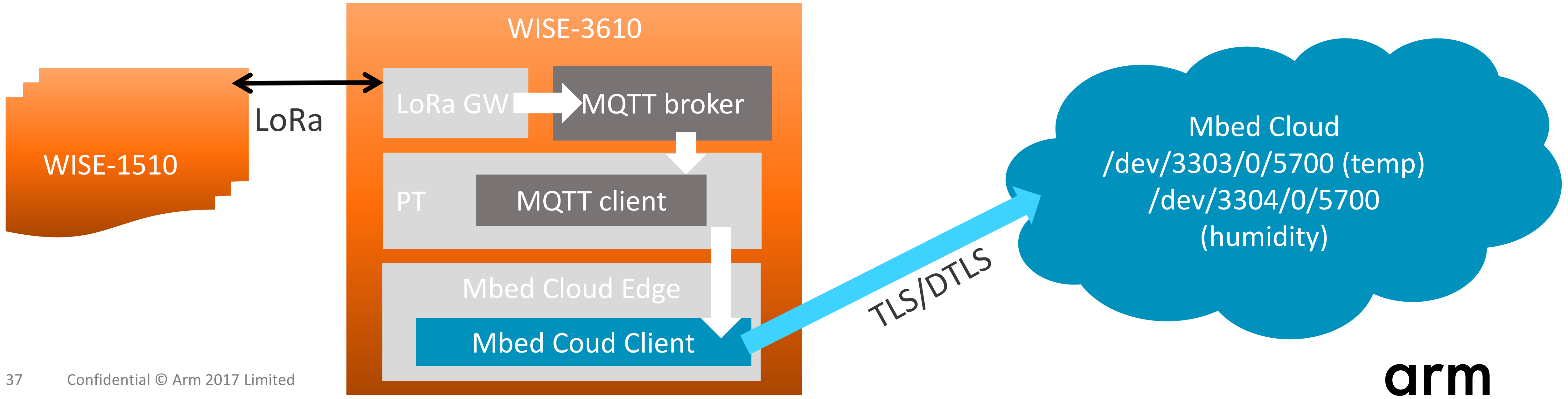
If device moves from one mbed Cloud Edge to another – what do you do with the event queue?

Demo – WISE 3610 with WISE 1510 LoRa end-point

Demo: WISE 3610 & WISE 1510

Jaakko Kukkohovi made an trial Protocol Translator for connecting the WISE 1510 to WISE 3610.

- This allows the WISE-1510 to present it's sensor data (temperature, humidity) in the mbed Cloud as LWM2M resources.
- In essence it's not a LoRa protocol translator but an **MQTT to LWM2M protocol** translator, because the WISE-1510s are visible in the Mosquitto service.



Visibility in the mbed Cloud REST API

Looking up the the mbed Cloud Edge & it’s devices

Via Postman or some other REST API Tool

Looking up the mbed Cloud Edge

You can look up the devices using the REST API call

https://api.us-east-1.mbedcloud.com/v3/devices/

You can then add a filter to it, which filters by the your device class for example

https://api.us-east-1.mbedcloud.com/v3/devices?filter=device_class%3DAdvantech_MBED_GW

Looking up the mbed Cloud Edge hosted devices

Edge hosted devices have a value in the field ” "host_gateway”.

You can find all Edge connected devices with:

https://api.us-east-1.mbedcloud.com/v3/devices/?filter=host_gateway__neq%3D%26%26 (i.e. host_gateway<>””)

You can find all specific Edge connected devices by matching the host_gateway to the Edge device ID.

https://api.us-east-1.mbedcloud.com/v3/devices/?filter=host_gateway%3D<edge_device_id>

Error scenarios, debugging

arm

Development tips

This is in essence typical Linux SW development – all the typical Linux development tools can be utilized.

- `valgrind` for memory leaks
- `gdb` for debugging (or any other tool you prefer that works with the toolchain).
- `strace` for system calls.

Enable the traces – current default trace levels is INFO, but you might want to have DEBUG.

`mbed_cloud_edge_config.h:#define MBED_TRACE_MAX_LEVEL TRACE_LEVEL_INFO` to
`mbed_cloud_edge_config.h:#define MBED_TRACE_MAX_LEVEL TRACE_LEVEL_DEBUG`

It does not compile – `toolchain.cmake`

The default `toolchain.cmake` we supply is generic - for running directly on a Linux machine that you are using.

For WISE-3610 (or any other SDK) – you must change the toolchain.

We have supplied another `toolchain.cmake` for the WISE-3610 via email – please copy that into folder `/build/mcc-linux-x86/` as `toolchain.cmake`.

More logs!

Enable the traces – current default trace levels is INFO, but you might want to have DEBUG.

`mbed_cloud_edge_config.h:#define MBED_TRACE_MAX_LEVEL TRACE_LEVEL_INFO` to
`mbed_cloud_edge_config.h:#define MBED_TRACE_MAX_LEVEL TRACE_LEVEL_DEBUG`

Did I get a connection?

Look in the serial logs of the mbed-core.

```
[INFO][edgecc]: on_registered_callback, registered 0 objects
[INFO][edgecc]: on_registered_callback, unregistered objects
count 0
```

On the callback invocation of the registered you know the mbed Cloud Client got it’s connection to the mbed Cloud.

If that does not show up, there is some problem.

Typical issues

Linux – can't connect.

- `[DBG ][PAL ]: Network failed reading interface info`
- Can be related to the IPv6 network i/f chosen, but IPv6 networking not working end-2-end.
- You can force IPv4 network by overriding the `PAL_NET_DNS_IP_SUPPORT` to 2 in the file `build/mcc-linux-x86/mbed_cloud_edge_config.h`.
  - I.e. add `#define PAL_NET_DNS_IP_SUPPORT 2` to that file.

Typical issues

- Developer certificate allows only 100 devices – you can easily run out of those.
  - You get error code quota exceeded if you have DEBUG level traces on.
  - Check the device directory for the amount of devices.
  - Delete the ones you don't need anymore (via the portal).

Linux – lots of "failed to create folder" errors in start.

-
```
[DBG ][PAL ]: Failed to create folder, was the storage properly initialized?
```
- Create the PAL folder for the KCM data. (`mkdir -p pal`).
  - If you use BYOC, you must have this folder in the device prior – so typically this only happens with developer mode.
  - This will be improved as well.

Typical issues

Linux – can’t connect

- `[ERR ][regc]: ConnectorClient::create_register_object - Failed to read credentials`
- Either:

1. Developer certificate is not found correctly. (Place the developer cert under `/client-wrapper` –folder).
2. If you chose BYOC mode, ensure the Factory Client injected the certificates to the same place where the Mbed Cloud Edge (or mbed Cloud Client for that matter) is trying to use it from.
   - By default the mbed Cloud Edge uses the `pal`folder in the smae folder as where the binary is.
   - Update example uses `/mnt/kcm`.

Thank You!
Danke!
Merci!
谢谢!
ありがとう!
Gracias!
Kiitos!