Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

# MEMORY USAGE OPTIMIZATION IN THE IMPLEMENTATION OF THE TR-069 PROTOCOL (CWMP) FOR MOCA MODEMS

Degree Final Work

**Degree in Computer Engineering**

**Author**: Manuel García Belmonte

**Tutor**: Lenin Guillermo Lemus Zúñiga

2020/2021

# Resumen

El presente trabajo versa sobre la reducción de memoria en los módems MoCA de la compañía MaxLinear que incorporan soporte para el protocolo TR-069 o CWMP. Este protocolo permite gestionar remotamente los dispositivos conectados a una red a través de servidores de autoconfiguración o ACS. Estos dispositivos cuentan con una reducida memoria RAM y se ha detectado que dicho protocolo realiza un gran consumo de recursos.

Con el fin de estudiar qué funciones están realizando un mayor consumo se ha puesto en marcha un simulador del dispositivo del cual se conocía su existencia pero no su estado ni cómo utilizarlo. Mediante la simulación se han llevado experimentos para conocer los consumos de memoria del cliente CWMP así como se ha desarrollado una propuesta que consiste en el envío fragmentado de los mensajes.

La mejora presentada en este proyecto ha conseguido reducir en un 61% el uso de memoria realizado por el protocolo TR-069 en una operativa habitual.

**Palabras clave:** memoria, CWMP, TR-069, módem, embebido

# Abstract

This work focuses on the memory reduction for MaxLinear's MoCA modems that features TR-069 or CWMP protocol support. This protocol allows to remotely manage the network connected devices through an auto configuration server or ACS. These devices have a small amount of RAM memory and previous studies have detected that this protocol needs high amounts of resources.

With the aim of studying which are the functions that have higher memory needs, a simulator has been used. Simulation allow us to launch several experiments to know memory consumptions and also helped with the development proposal that consists on a fragmented message sending.

The proposal developed in this project has reduced by 61% the amount of memory need by TR-069 protocol during a normal operation mode.

**Keywords :** memory, CWMP, TR-069, modem, embedded

# Acronyms

| Acronym | Definition |
|---|---|
| **ACS** | Auto Configuration Server. |
| **API** | Application Programming Interface |
| **CI** | Continuous Integration |
| **CPE** | Customer Premises Equipment. |
| **CWMP** | CPE Wan Management Protocol. |
| **DOM** | Data Object Model. |
| **GPA** | Get Parameter Attributes |
| **GPN** | Get Parameter Names |
| **GPV** | Get Parameter Values |
| **GPx** | Get Parameter [Attributes, Names, Values] |
| **MoCA** | Multimedia over Coaxial Alliance |
| **OS** | Operating System |
| **RPC** | Remote Procedure Call. |
| **RTOS** | Real Time Operating System |
| **R&D** | Research and Development |
| **SAX** | Simple API for XML |
| **SOAP** | Simple Object Access Protocol. |
| **SPV** | Set Parameter Values |
| **SPx** | Set Parameter [Attributes, Names, Values] |
| **TLS** | Transport Layer Security. |
| **TR69** | Technical Review 069 |
| **XML** | Extensible Markup Language |

# Table of contents

## List of figures

## List of tables

# 1. Introduction

Some studies estimate that 50.000 million devices are connected to the internet at the moment this project is being written [1]. The vast majority of them are embedded devices and share some peculiarities among them: their software and hardware are job-specific, meaning this that they have been engineered to perform an specific domain task -such as control the radiation of your everyday use microwave or even monitor critical variables when receiving radiotherapy during an oncological treatment-.

Although embedded systems significantly differ ones from others depending on their application, they usually address the same engineering challenges for the programmer: resources are limited due to several constraints, being memory one of the most critical parts of an embedded device.

This project has been developed during an internship in the American company MaxLinear. MaxLinear is a fabless semiconductors company specialized in solutions for access and connectivity with a R&D facility in Valencia.

## 1.1. Motivation

MaxLinear manufactures ICs that grant connectivity to millions of people around the globe. Some of those chips allow service providers to deploy modems that connect individuals to the Internet.

Having hundreds or even thousands of devices is not a big deal for big service provider companies such as AT&T, Verizon or Vodafone: they can have thousands of technicians that install, maintain and repair their systems. The issue arises when more than 4.600 million people are connected to the world through modems [2] and the Internet has become a first need: deploying and solving any potential issue cannot be done physically by a technician in an economical and temporal efficient way.

A new way of managing the devices was needed and here is where CWMP (CPE Wan Management Protocol) comes into action: the first technical report referring to this protocol was introduced by the Broadband Forum in May 2004 under the name of TR-069 and several amendments have been published since then [3]. This protocol allows service providers to remotely manage the devices deployed in customer premises.

As the firmware of the modem devices evolves with more features requested, the information managed by the CWMP protocol grows. This situation is not a problem when applications are programmed for general purpose computers but turns into a challenge when built for embedded devices as resources are difficult to enlarge -if not impossible- for the already deployed systems. And, of course, backward compatibility is a must.

I have always been attracted by embedded devices and used to realize small projects in my spare time since I started studying this degree. Having the opportunity to work in cutting edge projects and develop the knowledge acquired during the studies is what motivated this work.

This project addresses the issue of high memory consumption due to the usage of CWMP protocol in MoCA modems developed by company MaxLinear.

## 1.2 Current state

MoCA modems were initially developed by a company acquired by MaxLinear and transferred to the R&D Valencia group at the beginning of year 2020.

The solution was completed and fully functional at the moment I joined the company. The software was being tested by some important customers to reach its production phase by mid-2021.

Some recently added functionality requested by customers led to reducing the system free memory to as low as less than 1% of total memory. This compromised the future of new releases and features while impacting the performance of the modem as new memory needs will necessarily decrease the available memory for packet routing and other MoCA functionalities.

Biggest memory consumption was identified prior to my inclusion to the project: the CWMP protocol was the biggest consumer but a deeper study of the memory usage per function was still needed.

On the other hand, a simulator product was present in the code repository as a developing tool but was never used by the Valencia team and documentation was almost nonexistent. Current developers do not use the tool as it cannot be even compiled and there are no usage guides.

## 1.3. Goals

The main goal of this project is to optimize the memory usage by the protocol CWMP in MaxLinear’s MoCA modems. This purpose should be reached by the achievement of the following incremental points:

1. Compile and start the simulation product.
2. Detect where is taking place the higher memory consumption.
3. Implement a solution that reduces the memory usage of the system.
4. Test the solution in a real environment and define an optimized new memory layout.

# 2. TR-069 or CWMP protocol

TR-069 or CWMP (CPE WAN Management Protocol) is a technical specification published by the Broadband Forum that defines an application layer protocol that allows to remotely manage CPEs (customer-premises equipment) connected to an IP network.

The protocol is based on a bidirectional SOAP communication between the CPE and an ACS (Auto-Configuration Server) over HTTP.

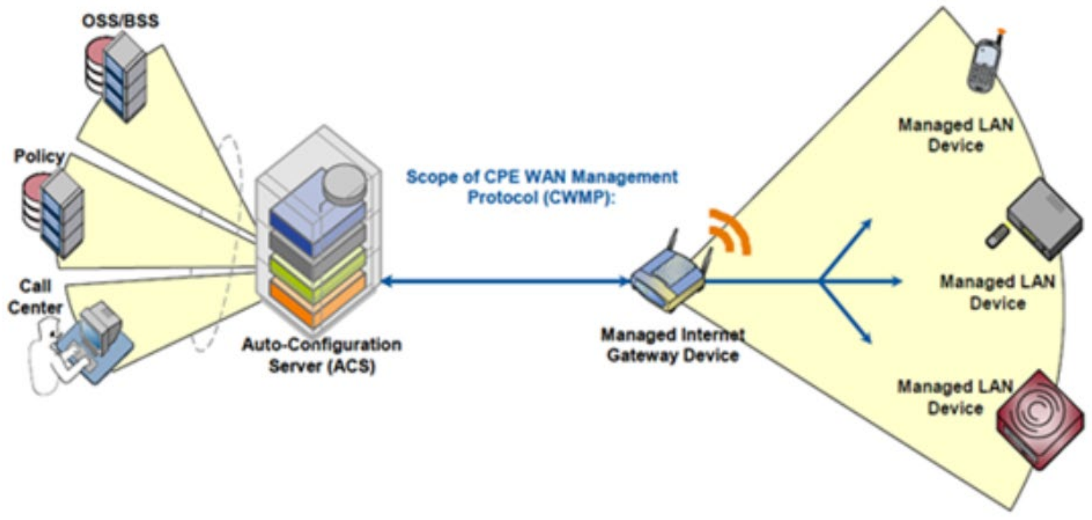

OSS/BSS
Policy
Call Center
Auto-Configuration Server (ACS)
Scope of CPE WAN Management Protocol (CWMP):
Managed Internet Gateway Device
Managed LAN Device
Managed LAN Device
Managed LAN Device

*Figure 1: typical architecture of a TR-069 session*
*Source: Remote Management of CableFree LTE CPEs using TR-069*
*https://www.cablefree.net/wirelesstechnology/4glte/lte-cpe-tr-069/*

SOAP is a messaging protocol that uses XML to exchange structured data. The XML document structure consists of different elements as described in its specification document [4]:

- The envelope: is mandatory and identifies the document as a SOAP message. TR-069 includes its own datamodel in the envelope.
- The header: is optional and includes some application-specific information about the message.
- The body: is mandatory and contains the call or response. It is actually the message to be sent. TR-069 uses the body to encapsulate its remote procedure calls (RPCs).
- The fault: is optional and indicates error messages.

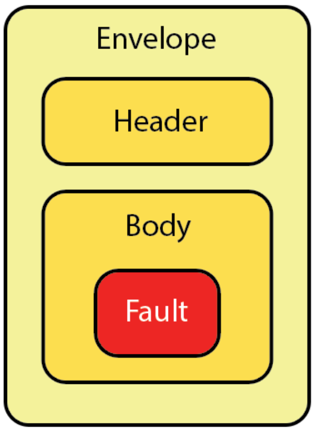
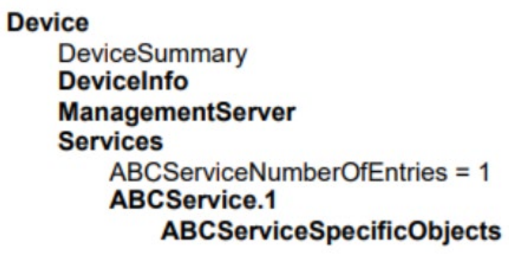


*Figure 2: SOAP message structure*

By using TR-069 protocol, ACS servers are able to remotely call the available RPCs implemented by the devices (CPE) in order to monitor, diagnose and even upgrade their firmware.

A series of datamodels are defined by the Broadband Forum to be used by CWMP protocol. This way, TR-069 enabled devices always have a root datamodel called Device that stablishes a series of common objects and components that can the instantiated by the device and describe its information such as interfaces, software version, parameters, etc. Moreover, specific datamodels are defined to be used in particular applications such as VoIP services.

The ACS usually requests the whole device datamodel to know the capabilities of the device and how to manage it. This is done sending Get Parameter Attributes (GPA), Get Parameter Names (GPN) and Get Parameter Values (GPV) RPCs. The server can also modify some of the values by calling their respective setter functions: SPA and SPV.



```
Device
    DeviceSummary
    DeviceInfo
    ManagementServer
    Services
        ABCServiceNumberOfEntries = 1
        ABCService.1
            ABCServiceSpecificObjects
```

*Figure 3: TR-069 datamodel hierarchy example*

## 2.1. A typical TR-069 session

A quick overview of a typical TR-069 session will help to understand the current implementation of the protocol in MaxLinear MoCA modems:

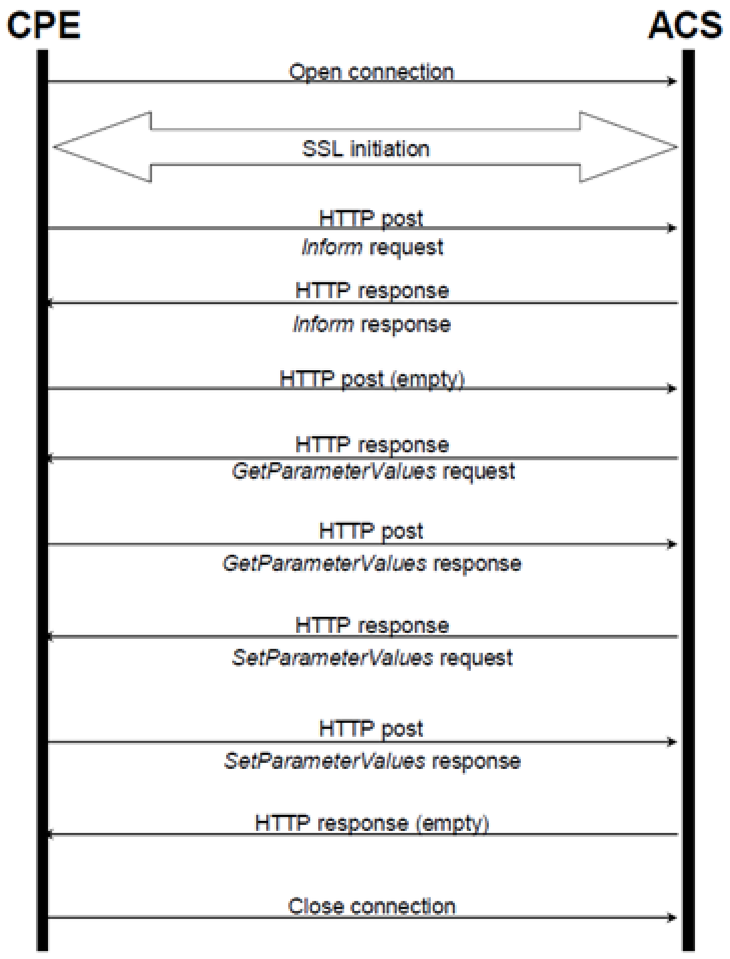


*Figure 4: typical TR-069 session*
*Source: CWMP communications*
*https://commons.wikimedia.org/wiki/File:ComunicacionTR69_1.JPG*

1. The CPE opens the connection and optionally
2. A secure connection is negotiated between both devices.
3. The CPE starts the session sending an Inform Request over an HTTP post. It contains some basic information such as the device ID or the events that caused the CPE to start the session.
4. The ACS answers with an Inform Response, confirming that the Inform has been correctly processed.

5. If the CPE doesn’t have more information to deliver to the ACS, it sends an empty HTTP Post meaning “no more work”.
6. If the ACS has RPCs to be sent, sends the appropriate HTTP response with the method requested. If not, sends an empty HTTP response (step 10).
7. The CPE delivers the response with the data the ACS was looking for in an HTTP Post.
8. The ACS could need to set some values and sends a Set Parameter request.
9. The CPE returns a response indicating if the changes the ACS requested have been performed or not.
10. If the ACS has no more RPCs to send to the CPE, it sends an empty HTTP response.

11. The CPE closes the session.

As seen, in TR-069 the CPE always initiates the session. A periodic inform is set to constantly initiate communication with the server. Nevertheless, in some cases the ACS needs to perform some RPCs that cannot wait until the periodic inform is triggered. In this case, a Connection Request is sent to force the CPE to start the session.

Due to today’s network architectures, CPEs are not accessible if they are behind a NAT gateway. Some mechanisms are used to reach the CPE (such as STUN or XMPP [5]) but are out of the scope of this document.

## 2.2. Current implementation

Figure 5 describes the current implementation of the TR-069 client in MaxLinear MoCA modems.

We can mainly identify three .c files involved:

- tr69_client.cwmp.c: CWMP client *main* function is located in this file. It is in charge of stablishing the connection when the device boots up and when the periodic inform event is received. Receives the messages following the flow showed in Figure 4. Also encapsulates the response in a SOAP message and sends it.

- tr69_client_soap.c: is responsible of identifying the remote procedure call (RPC) requested by the ACS. Once identified, calls the appropiate device implementation of the RPC. Finally gets the parameter list retrieved by the Glue Layer and builds the XML message.
- tr69_glue_api.c: it is an API called Glue Layer that translate RPCs to a specific device function that retrieves the information requested. It is architecture dependent.

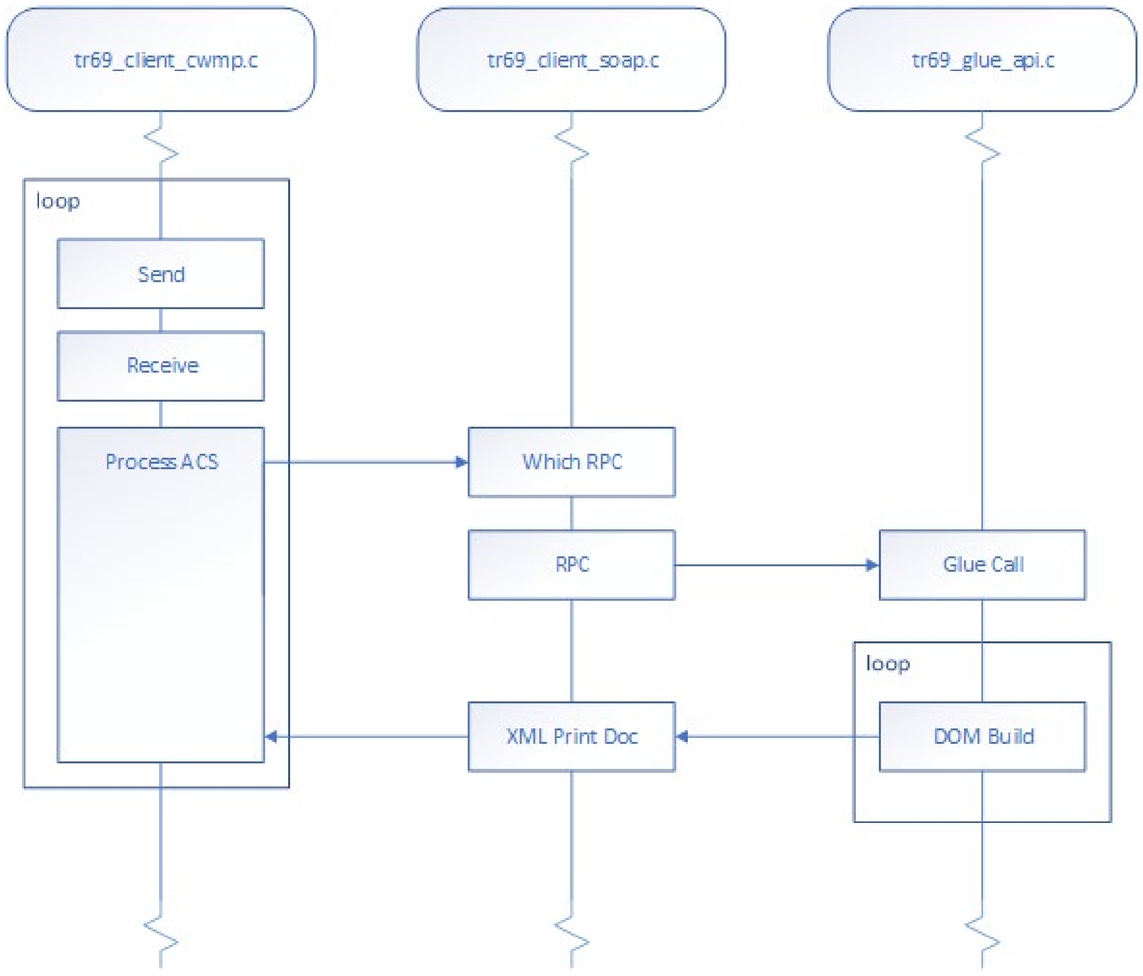


*Figure 5: MoCA TR-069 client sequence diagram*

The remote procedure calls supported by our device are the following:

- GetRPCMethods
- GetParameterAttributes
- SetParameterAttributes
- GetParameterNames

- GetParameterValue
- SetParameterValue
- AddObject
- DeleteObject
- Reboot
- Download
- FactoryReset

When the ACS requests a particular RPC, a specific glue API function is called. Get Parameter Attributes, Names and Values are the RPCs called when the ACS needs to retrieve information about the current instantiation of the device data model. Therefore, specific functions from the glue layer are called to get all the information.

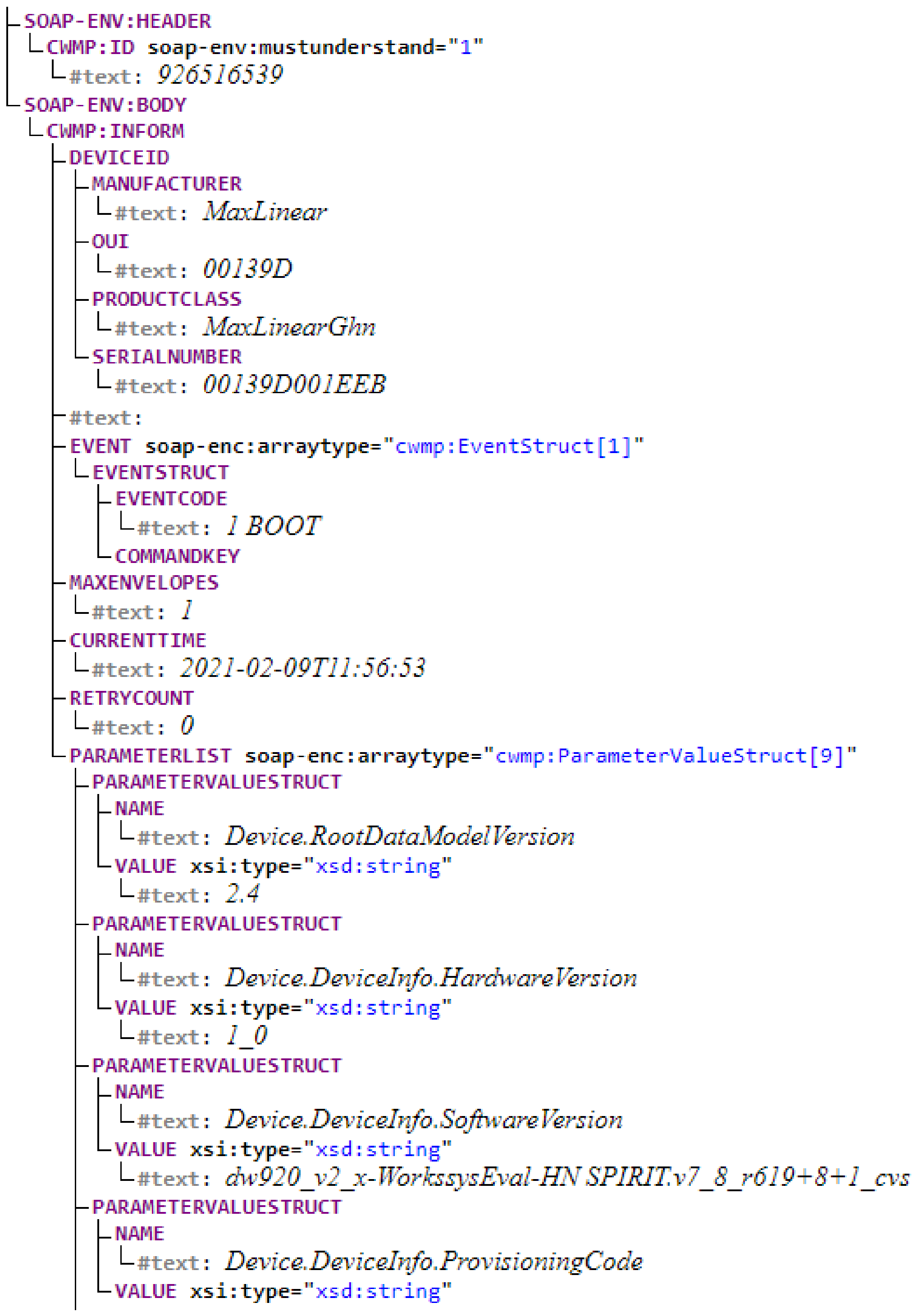
Glue Layer returns the parameters in a linked list. As messages need to be sent in XML format, each RPC function builds a DOM document containing all the requested parameters.

## 2.3. Document Object Model (DOM)

Data Object Model or DOM is a tree representation of an XML or HTML document [6]. Representing an XML document in the form of nodes allow us to navigate through the data to retrieve and modify the needed object.

MoCA firmware uses an Intel GPL library that manages DOM documents and allows a fast creation and modification of the nodes.

Figure 6 shows a piece of an Inform message modelled in a DOM document.



```
├SOAP-ENV:HEADER
│└CWMP:ID soap-env:mustunderstand="1"
│  └#text: 926516539
└SOAP-ENV:BODY
  └CWMP:INFORM
    ├DEVICEID
    │├MANUFACTURER
    ││└#text: MaxLinear
    │├OUI
    ││└#text: 00139D
    │├PRODUCTCLASS
    ││└#text: MaxLinearGhn
    │└SERIALNUMBER
    │  └#text: 00139D001EEB
    ├#text:
    ├EVENT soap-enc:arraytype="cwmp:EventStruct[1]"
    │└EVENTSTRUCT
    │  ├EVENTCODE
    │  │└#text: 1 BOOT
    │  └COMMANDKEY
    ├MAXENVELOPES
    │└#text: 1
    ├CURRENTTIME
    │└#text: 2021-02-09T11:56:53
    ├RETRYCOUNT
    │└#text: 0
    └PARAMETERLIST soap-enc:arraytype="cwmp:ParameterValueStruct[9]"
      ├PARAMETERVALUESTRUCT
      │├NAME
      ││└#text: Device.RootDataModelVersion
      │└VALUE xsi:type="xsd:string"
      │  └#text: 2.4
      ├PARAMETERVALUESTRUCT
      │├NAME
      ││└#text: Device.DeviceInfo.HardwareVersion
      │└VALUE xsi:type="xsd:string"
      │  └#text: 1_0
      ├PARAMETERVALUESTRUCT
      │├NAME
      ││└#text: Device.DeviceInfo.SoftwareVersion
      │└VALUE xsi:type="xsd:string"
      │  └#text: dw920_v2_x-WorkssysEval-HN SPIRIT.v7_8_r619+8+1_cvs
      ├PARAMETERVALUESTRUCT
      │├NAME
      ││└#text: Device.DeviceInfo.ProvisioningCode
      │└VALUE xsi:type="xsd:string"
      │
```

*Figure 6: DOM representation of an Inform message*

# 3. Methodology

## 3.1. Organization

The MoCA development team is composed of five people: three developers based in Valencia, one in Taiwan and a manager in Valencia who also participates in developing tasks. On the other hand, the software quality assurance team (SQA) is composed of three people: a developer and manager in Valencia and a third developer based in Carlsbad, California. Both managers report to their respective site managers (firmware and SQA) who in turns report to the international business area manager.

The company uses agile methodologies to organize the work of every component of the team. Tasks are defined in 2 week duration sprints with 1 hour weekly sync-up meetings in which every member report the progress on their tasks of the past week and the expectations for the following one. If a team or group of developers need to go deeper into a specific topic and it doesn't make sense for the rest of attendants, an ad-hoc meeting is arranged.

Meetings are held online using Teams due to the current pandemic situation derived from COVID-19. Nevertheless, as the teams involved in a project are usually located not only in local but in any of the company offices around the world, videoconference tools were already used before.

Managers usually assign the tasks to every developer. Nevertheless, sometimes is the developer who suggests the tasks in which he or she should work in the following weeks. Developers have more visibility and understanding about the job they are doing and that is taken into consideration when it comes to task assignments. For this reason, developers are also in charge of estimating the duration of every task. The maximum task duration should not exceed 3 days (24h). If any task is supposed to need more than three days, it should be decomposed in smaller ones.

JIRA is used as a task management tool. This allows an integral management of every task in a project's context. At the same time, it's used as a knowledge database as every task a developer is working in must have been previously created in JIRA with its corresponding description and time estimates.

During a typical 2-week sprint, each team component must load tasks with a total duration of 8 days. The remaining 2 days are supposed to be needed in order to attend programmed and ad-hoc meetings, write and read emails, unforeseen issues, etc.

## 3.2. Development

Developers use GIT as a version control system that tracks changes in code. Different projects are organized into Bitbucket repositories. JIRA and Bitbucket are synchronized between them so when a developer creates a commit with his changes, the commit name must include the JIRA identifier. This way, JIRA is used as a knowledge database not only tracking all the issues but also containing the changes associated to every task.

Furthermore, an exhaustive process takes place before any changes are uploaded to the repository: once code changes are tested and the developer is sure that works as expected, a “code review” is created. At least three developers must join the code review so they can inspect the code and add some comments, questions or suggestions. Only once every doubt or modification is resolved, an approval is granted by every code reviewer. The developer can then merge his changes.

Regarding compilation, several Docker machines are deployed so the developer just needs to connect to them through SSH. This machines are properly configured to satisfy the dependencies needed for the compilation process. Using this technology allow the software team to easily maintain the infrastructure, offering to every developer the same environment ready from day one.

## 3.3. Laboratories and tests

The devices, modems, are organized in different test benches located in MaxLinear’s R&D centers. In our MoCA particular case, benches are located in Valencia and Carlsbad, California. As some of the devices are inaccessible to the developer, they are remotely managed through automatic switches that allow us to switch them on and off in case of need. They are also connected through serial to computers that can be accessed via web.

Benches are used by developers to test their changes in controlled environments. They allow us to reproduce customer setups. They also allow SQA team to verify that the firmware released by the firmware team passes internal tests and standard certifications (in our case, MoCA Alliance certification tests) prior to deliver a version to a customer.

As an automation tool for testing purposes, Jenkins is used. With this tool, SQA designs an automated flow based on continued integration or CI. Jenkins also allow us to manually book benches during a desired amount of time to avoid several developers to try to use the same bench concurrently or even avoid interfering with automated tests.

Jenkins tool is in charge of automatically launch the process of build: before some changes are merged to a branch, several automatic tests will be launched in an unattended way to verify that the new version doesn’t cause the modems to stop working. If tests pass, Jenkins will automatically build the different product images and place them in the selected output directory. Automated tests will use these binaries to upgrade the modems and perform the requested tests.

# 4. Memory Analysis

Previous analysis about the CWMP implementation showed some hints about where the higher memory consumption could be taking place. DOM representation of the response generated by some RPC calls that request the whole datamodel tree – or at least those that request many parameters- could be the cause of the higher memory consumption. Nevertheless, a deeper analysis is needed to determine exactly which functions are involved.

## 4.1. Current memory layout

Before starting with the memory analysis of each function and RPC, it is necessary to clarify how is the memory structured in the MoCA device and the mechanisms used to allocate and free them.

C language, in which the firmware is developed, provide us a malloc function to dynamically allocate memory. However, this mechanism is dangerous when used in embedded systems where the memory is usually a very limited resource. Malloc and free calls can fragment the RAM memory of the device. Fragmentation, with an absence of mechanisms like memory management in many OS, can lead to the impossibility to allocate the requested amount of contiguous memory even if the whole system has enough. This behavior can cause the device to stop working properly or even cause a crash, both undesirable situations.

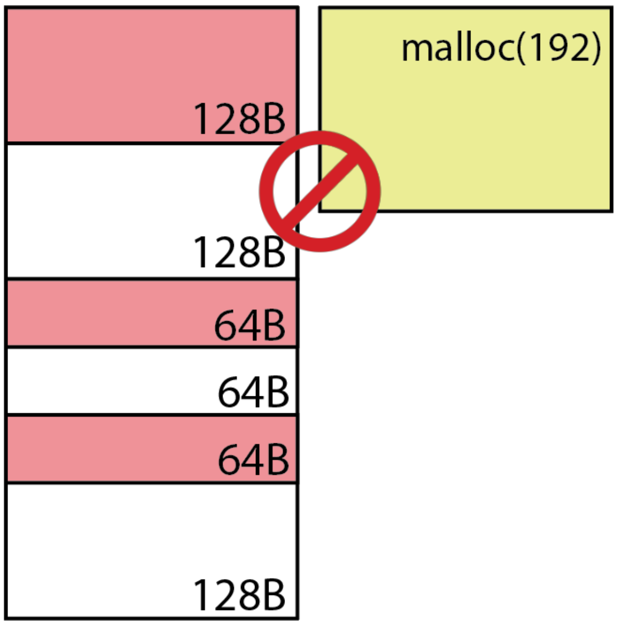


*Figure 7: example of fail allocation due to memory fragmentation*

The approach followed in our MoCA device firmware to get around this potential problem is based on defining a fixed structure of blocks of different sizes that will be allocated at boot time. New malloc and free wrappers will search for the smallest block available that satisfies the call. A pointer to a larger block can be returned if smallest blocks are already in use.

This approach reduces the fragmentation issue as memory is no longer allocated during runtime. However, a careful study of memory consumption of each part of the system is needed in order to build a layout as much accurate as possible. Otherwise, if not optimized, memory will be wasted assigning larger blocks than needed.

The fixed-size memory management mechanism is useful when applied to predictable environments [7] such as our TR-069 client.

Table 1 shows the memory layout when I started working on this project.

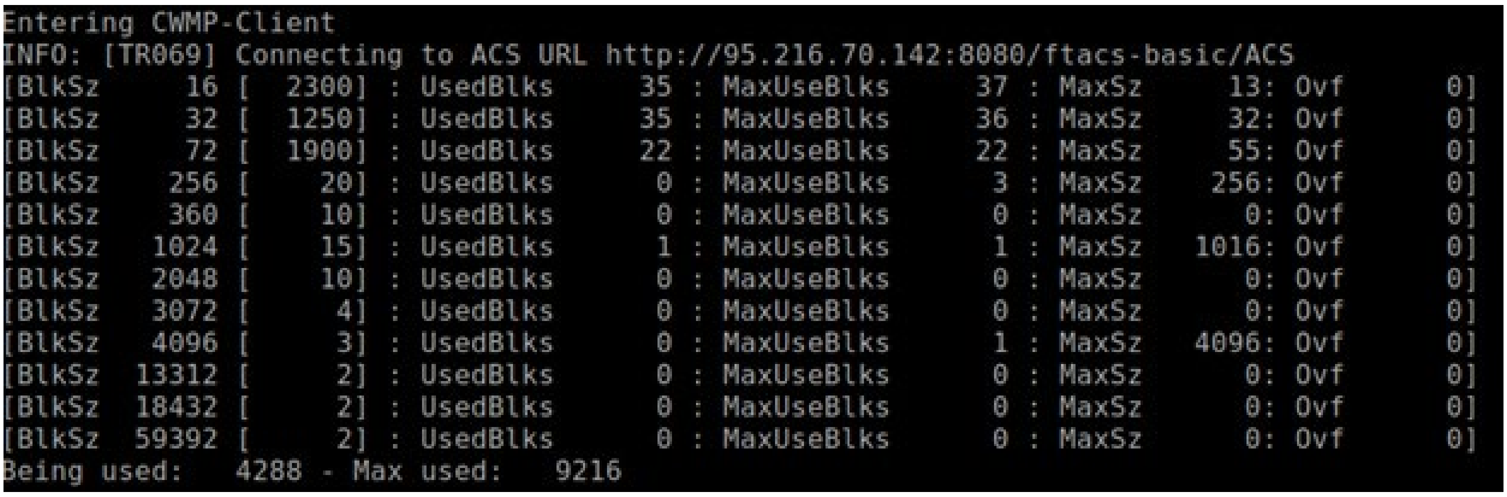
| Block size (Bytes) | Number of blocks |
|---|---|
| 16 | 2300 |
| 32 | 1250 |
| 72 | 1900 |
| 256 | 20 |
| 360 | 10 |
| 1024 | 15 |
| 2048 | 10 |
| 3072 | 4 |
| 4096 | 3 |
| 13312 | 2 |
| 18432 | 2 |
| 59392 | 2 |

*Table 1: previous memory layout*

With the aim of analyzing the memory usage of different CWMP client functions, a simple but useful function has been developed. HMEM_getMemUsage prints the following information:

- Block size.
- Number of blocks.
- Number of used blocks.
- Maximum number of used blocks since boot.
- Maximum requested size (value of malloc argument).

- Number of blocks overflow.



```
Entering CWMP-Client
INFO: [TR069] Connecting to ACS URL http://95.216.70.142:8080/ftacs-basic/ACS
[BlkSz      16 [  2300] : UsedBlks     35 : MaxUseBlks     37 : MaxSz     13: Ovf       0]
[BlkSz      32 [  1250] : UsedBlks     35 : MaxUseBlks     36 : MaxSz     32: Ovf       0]
[BlkSz      72 [  1900] : UsedBlks     22 : MaxUseBlks     22 : MaxSz     55: Ovf       0]
[BlkSz     256 [    20] : UsedBlks      0 : MaxUseBlks      3 : MaxSz    256: Ovf       0]
[BlkSz     360 [    10] : UsedBlks      0 : MaxUseBlks      0 : MaxSz      0: Ovf       0]
[BlkSz    1024 [    15] : UsedBlks      1 : MaxUseBlks      1 : MaxSz   1016: Ovf       0]
[BlkSz    2048 [    10] : UsedBlks      0 : MaxUseBlks      0 : MaxSz      0: Ovf       0]
[BlkSz    3072 [     4] : UsedBlks      0 : MaxUseBlks      0 : MaxSz      0: Ovf       0]
[BlkSz    4096 [     3] : UsedBlks      0 : MaxUseBlks      1 : MaxSz   4096: Ovf       0]
[BlkSz   13312 [     2] : UsedBlks      0 : MaxUseBlks      0 : MaxSz      0: Ovf       0]
[BlkSz   18432 [     2] : UsedBlks      0 : MaxUseBlks      0 : MaxSz      0: Ovf       0]
[BlkSz   59392 [     2] : UsedBlks      0 : MaxUseBlks      0 : MaxSz      0: Ovf       0]
Being used:   4288 - Max used:   9216
```

*Figure 8: HSMEM_getMemUsage output example*

Analyzing the first line output of HSMEM_getMemUsage showed in Figure 8, a total of 2300 blocks of 16B are allocated at boot time. At the time of the function call, 35 of those 16B blocks were taken while 37 blocks was the maximum number of 16B served by the malloc wrapper since the system booted.

“MaxSz” 13 states that the biggest block needed was indeed 13B but as there is not a smaller one, a 16B block was returned instead. If” Ovf” was bigger than “0”, it would indicate that an allocation of 16B was requested by could not been satisfied because all of them were already in use.

## 4.3. Functions to analyze

The strategy used to detect where is taking place the higher memory consumption is based on identifying some suspicious points in CWMP client and using HSMEM_getMemUsage function described previously to get the memory information at a given time.

Points identified to be analyzed are:

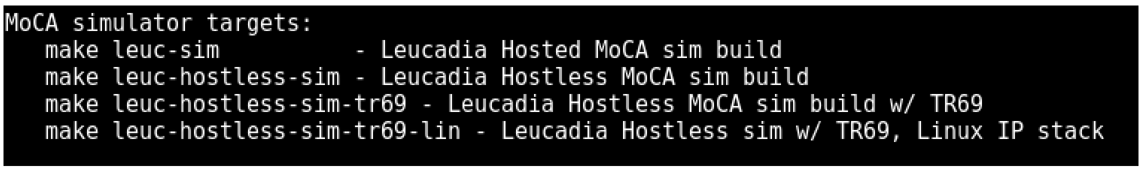
- Receiving the message: RPCs sent by the ACS are received by TR69_SoapRecv function.

- Message decoding: TR69_ProcessACSReq function is in charge of decoding the received message and call the appropiate function corresponding to the requested RPC.
- RPC call: each RPC that can be requested by the ACS has its corresponding function in the CWMP client. Although our CPE implements all functions listed in page XX, we selected the study of those which request or modify many parameters and could have bigger memory needs:
    - GetParameterAttributes.
    - GetParameterNames.
    - GetParameterValues.
    - SetParameterAttributes.
    - SetParameterValues.
    - GetRPC.
- XML message build: once processed the requested RPC, an XML message is built and sent to the ACS. ixmlPrintDocument is the function in charge or parsing the DOM document and building the XML message.

## 4.4. Testing setup

The main objective of this project, as stated earlier, was the memory usage reduction in the CWMP protocol implementation in MaxLinear's MoCA modems. Nevertheless, the existence of a simulation product was known by the team but the Valencia team didn't have expertise on this. Its state was unknown so they didn't know if it was operative and how to use it. Setting up the simulator was one of the sub-objectives of the project. It would help the memory analysis task while could become a useful tool for the whole team in further developments.

The Makefile of the project showed different simulation products as shown in Figure 9. We decided to focus on the product that included the TR-069 protocol and the Linux IP stack.



```
MoCA simulator targets:
   make leuc-sim           - Leucadia Hosted MoCA sim build
   make leuc-hostless-sim - Leucadia Hostless MoCA sim build
   make leuc-hostless-sim-tr69 - Leucadia Hostless MoCA sim build w/ TR69
   make leuc-hostless-sim-tr69-lin - Leucadia Hostless sim w/ TR69, Linux IP stack
```

*Figure 9: MoCA simulation targets*

A Linux environment we set up using a virtual machine so we can execute the binary built after leuc-hostless-sim-tr69-lin product compilation. Prior to that, it was necessary to solve some issues related to the compilation as some libraries were missing. To accomplish this task, a better understanding about how the simulator was coded was necessary. The code that was present in the repository was full of sentences like the following one:

```
#if defined(SIMULATOR)
#include “library.h”
#endif
```

This compiler directives allow not only to include the libraries needed for the simulation product to compile but also allow the developer to split a function that differs from its implementation for different compilation products, avoiding the need of having different files for those products. This approach is used, for instance, in the IP stack implementation and thread management: the simulation needs a different piece of code than the real hardware but both are coded in the same .c file and function name.

Once the product compiled successfully, it was necessary to modify the Linux environment in order to make it work:

- Add 32-bit architecture compatibility: the simulation application thrown a compatibility error due to is 32-bit architecture. As the Linux machine was 64-bit, 32-bit compatibility was needed to make the OS capable of executing it.
- TAP interface creation: once the compatibility issue was solved, an error related with an absence of tap interface was shown. Tap interfaces are offered by the Linux kernel and basically are software interfaces without any hardware component related to them. This interface allow to send and receive packets to an application –the simulator in our case- that is attached to it. The advantage of using tap interfaces lies in the simplicity of capturing the packets that go through them by using an appropriate tool such as Wireshark.

  Openvpn tool was used for that purpose with the following commands:

  ```
  openvpn --mktun --dev tap0
  ip link set tap0 up
  ip addr add 10.0.0.1/24 dev tap0
  ```

- Set a couple of processor in the virtual machine: once the earlier issues were solved, the application started its execution but showed an error related with pthread library. A deeper study of the simulator code led us to notice that POSIX threads were used to simulate the behavior of threads created in real hardware RTOS. The simulation application needed the OS to be configured with at least 2 processors. Therefore, it was needed to configure VirtualBox in such way.

Once the environment was properly set up and the behavior of the simulator was understood, we decided to continue using the simulator product during the following phases of the project: analyzing memory usage and implementing a reduction proposal.

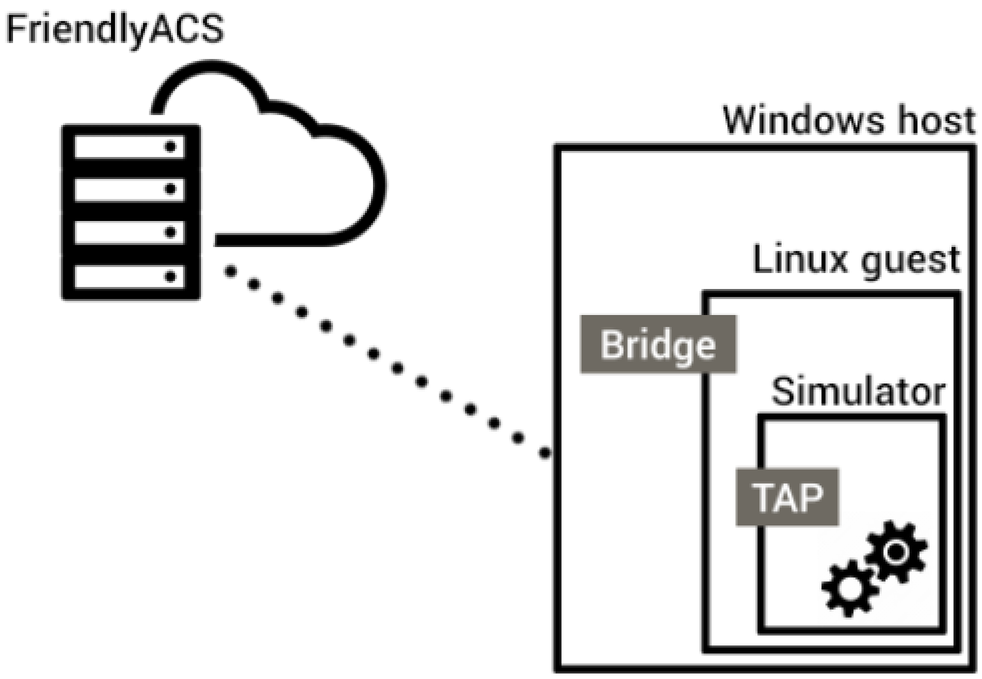


*Figure 10: testing environment setup*

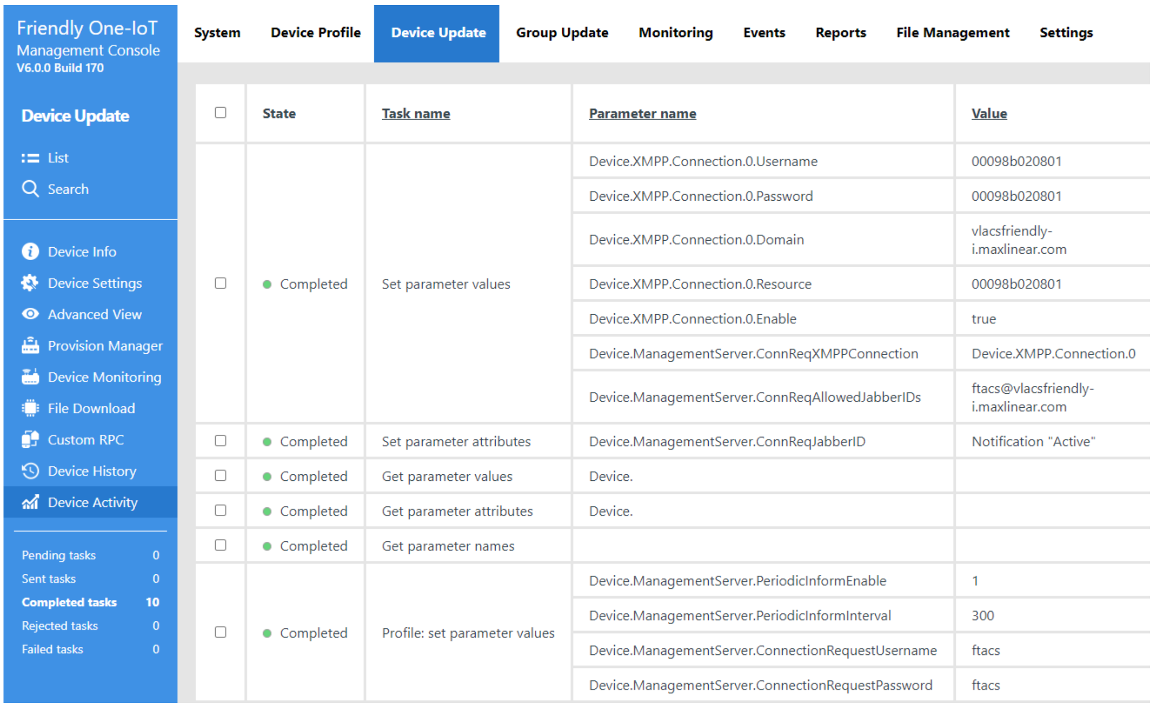
The use of this simulator allows to boost the process of analysis and implementation since:

1. Compilation is faster: production firmware for real modems needs to use specific platform compiler which also requires to connect to a license server located in the US, increasing time to compile.

2. Loading the firmware into the device is not necessary when using the simulator. Real modems need to be upgraded through serial port which is a slow process. They also need to be reflashed when the firmware contains some bug and the device crashes. Simulator is launched in a bash terminal so executions are a lot faster.
3. Analyzing packets is easier as they can be inspected using an analyzing tool such as Wireshark. Tap interfaces allow to inspect full Ethernet frames avoiding the need of additional hardware to capture them.

Figure 9 shows the final setup used to perform the memory analysis showed in the following chapter. We can observe how the simulation application uses the TAP interface to communicate with the guest OS. The virtual machine uses a bridge with the host OS so it can reach the outside.

MaxLinear has access to a cloud-hosted ACS server that will be used to manage the CPEs. Using its administration console via web, different predefined and custom RPCs can be sent to the devices. It also shows valuable information about the RPC result: whether they succeeded or not. Figure 11 shows a screenshot of Friendly ACS GUI.



| | |
|---|---|
| Pending tasks | 0 |
| Sent tasks | 0 |
| **Completed tasks** | **10** |
| Rejected tasks | 0 |
| Failed tasks | 0 |

| ☐ | State | Task name | Parameter name | Value |
|---|---|---|---|---|
| ☐ | ● Completed | Set parameter values | Device.XMPP.Connection.0.Username | 00098b020801 |
| | | | Device.XMPP.Connection.0.Password | 00098b020801 |
| | | | Device.XMPP.Connection.0.Domain | vlacsfriendly-i.maxlinear.com |
| | | | Device.XMPP.Connection.0.Resource | 00098b020801 |
| | | | Device.XMPP.Connection.0.Enable | true |
| | | | Device.ManagementServer.ConnReqXMPPConnection | Device.XMPP.Connection.0 |
| | | | Device.ManagementServer.ConnReqAllowedJabberIDs | ftacs@vlacsfriendly-i.maxlinear.com |
| ☐ | ● Completed | Set parameter attributes | Device.ManagementServer.ConnReqJabberID | Notification "Active" |
| ☐ | ● Completed | Get parameter values | Device. | |
| ☐ | ● Completed | Get parameter attributes | Device. | |
| ☐ | ● Completed | Get parameter names | | |
| ☐ | ● Completed | Profile: set parameter values | Device.ManagementServer.PeriodicInformEnable | 1 |
| | | | Device.ManagementServer.PeriodicInformInterval | 300 |
| | | | Device.ManagementServer.ConnectionRequestUsername | ftacs |
| | | | Device.ManagementServer.ConnectionRequestPassword | ftacs |

*Figure 11: Friendly ACS GUI*

## 4.5. Memory usage analysis per function

The following piece of code shows the memory consumption procedure. In this particular case, the reception of the message is taken as example but the method is analogue to all measurements made through this chapter.

```
TR69_Free(msg);
msg = NULL;
HSMEM_getMemUsage();
TR69_Receive(msg);
HSMEM_getMemUsage();
```

The call to HSMEM_getMemUsage() prints on console the current memory block state, as was shown in Figure 8. The information is printed twice: once prior the function call to analyze and lastly once the function has returned.

The difference between both time instants shows the information desired: the memory allocation that took place during the function call to analyze.

### 4.5.1. Reception of the message

To study the reception of the message memory consumption, different RPCs were selected: GPA, GPN and GPV of full tree is a common RPC sent by the ACS to register the CPE and also to get a provision. Once the device is registered, the ACS usually sends some Set Parameters RPC. It usually sets from 1 to 4 parameter but we decided to test up to 10 parameters. During normal operation, the ACS requests some parameters so a GPA, GPN and GPV of 10 parameters is also measured. Finally, an Inform Response and GetRPC are also analyzed.

| RPC | Number of parameters | Memory consumption (kB) |
|---|---|---|
| **GPA/GPN/GPV** | 1 (full tree) | 1.0 |
| **GPA** | 10 | 2.0 |
| **GPN** | 10 | 2.0 |
| **GPV** | 10 | 2.0 |
| **SPA** | 1 | 1.0 |
| **SPA** | 4 | 2.0 |
| **SPA** | 10 | 3.0 |
| **SPV** | 1 | 1.0 |
| **SPV** | 4 | 2.0 |
| **SPV** | 10 | 3.0 |
| **Inform Response** | - | 1.0 |
| **Get RPC** | - | 1.0 |

*Table 2: message of message memory consumption*

Analysis: memory usage did not exceed 3kB in any of the measurements. As expected, those RPC that contain more parameters need largest amounts of memory: the buffer needed to store the XML received message is bigger. However, RPCs with up to 10 parameters do not seem to compromise the system even though using 10 parameters is not a common behavior. GPA, GPN and GPV of full tree, which involves the largest number of parameters, just need 1kB as the message needed to request the whole datamodel only needs one parameter, the root path.

### 4.5.2. Message decoding

A similar set of RPCs were chosen to analyze the message decoding. Inform Response has not been tested as this is not the kind of message that triggers the decoding function.

| RPC | Number of parameters | Memory consumption (kB) |
|---|---:|---:|
| **GPA/GPN/GPV** | 1 (full tree) | 3.1 |
| **GPA** | 10 | 5.6 |
| **GPN** | 10 | 5.6 |
| **GPV** | 10 | 5.6 |
| **SPA** | 1 | 4.3 |
| **SPA** | 10 | 28.9 |
| **SPV** | 1 | 3.9 |
| **SPV** | 4 | 21.7 |
| **SPV** | 10 | 24.3 |
| **Get RPC** | - | 2.7 |

*Table 3: message decoding memory consumption*

Analysis: message decoding involves parsing the XML received so the memory usage increases as the message itself does: the more parameters to get or set, the more memory needed. Nevertheless, a maximum of ~30kB is observed for a SPA with 10 parameters therefore the decoding function does not seem to be the bottleneck of the system.

### 4.5.3. Specific RPC calls

To study the specific device implementation of each remote procedure call, the following RPC were selected: GPA, GPN and GPV of full tree and 10 parameters. SPA and SPV of 1 and 10 parameters. Get RPC was also studied.

| RPC | Number of parameters | Memory consumption (kB) |
|---|---|---|
| GPA | 1 (full tree) | 202.15 |
| GPN | 1 (full tree) | 126.4 |
| GPV | 1 (full tree) | 148.1 |
| GPA | 10 | 17.0 |
| GPN | 10 | 6.9 |
| GPV | 10 | 8.1 |
| SPA | 1 | - |
| SPA | 10 | 14.7 |
| SPV | 1 | 22.9 |
| SPV | 10 | 18.3 |
| Get RPC | - | 2.8 |

*Table 4: specific RPC calls memory consumption*

Analysis: this experiment shows the higher memory consumption until now. Biggest memory allocations are taking place in GPA, GPN and GPV calls for the whole tree. This confirmed the hypothesis that full tree requests were the cause of the biggest memory needs as the resulting message is the largest one. It also showed that GPA needs 36% more memory than GPV, the second biggest consumer.

Setter functions memory needs increases with the number of parameters but this behavior does not seem to be an issue as setting up to 10 parameters

### 4.5.4. XML message construction

Finally, the function that builds the XML message was analyzed. GPA, GPN and GPV of full tree and 10 parameters were sent. GetRPC and Set Parameter RPCs were also analyzed for 1, 4 and 10 parameters although setter functions just reply with a OK/KO response so we expected small memory needs.

| RPC | Number of parameters | Memory consumption (kB) |
|---|---|---|
| **GPA** | 1 (full tree) | 222.8 |
| **GPN** | 1 (full tree) | 220.1 |
| **GPV** | 1 (full tree) | 226.4 |
| **GPA** | 10 | 17.1 |
| **GPN** | 10 | 16.0 |
| **GPV** | 10 | 15.7 |
| **SPA** | 1 | 1.0 |
| **SPA** | 10 | 1.0 |
| **SPV** | 1 | 1.0 |
| **SPV** | 4 | 1.0 |
| **SPV** | 10 | 1.0 |
| **Get RPC** | - | 13.2 |

*Table 5: XML message construction memory consumption*

Analysis: high amounts of memory are also requested by this message building function. Around 220kB are allocated for the three GPA, GPN and GPV functions when the whole tree is requested. This time there are no big differences between them as showed in specific RPC function analysis. Setter RPCs memory usage was 1kB even on those with 10 parameters. This was the expected behavior as the arguments they have when called do not affect the response.

## 4.6. Preliminary findings

In the light of the results that different measurements of RPCs show, we can conclude that calls to getter functions (GPA, GPN and GPV) are those that need biggest memory to perform its task when the full tree is requested. High allocations are taking place not only during the RPC function itself but also during the response message (XML) build.

Memory allocation also grows at the message reception and decoding when large getter and setter RPCs are called. However, this has not been considered a dangerous behavior as number of parameter requested do not usually exceed from four.

In any case, a small study increasing the number of parameters requested by a setter RPC was done to have more visibility about when a large SPN or SPV could compromise system stability and is showed in Figure 10.

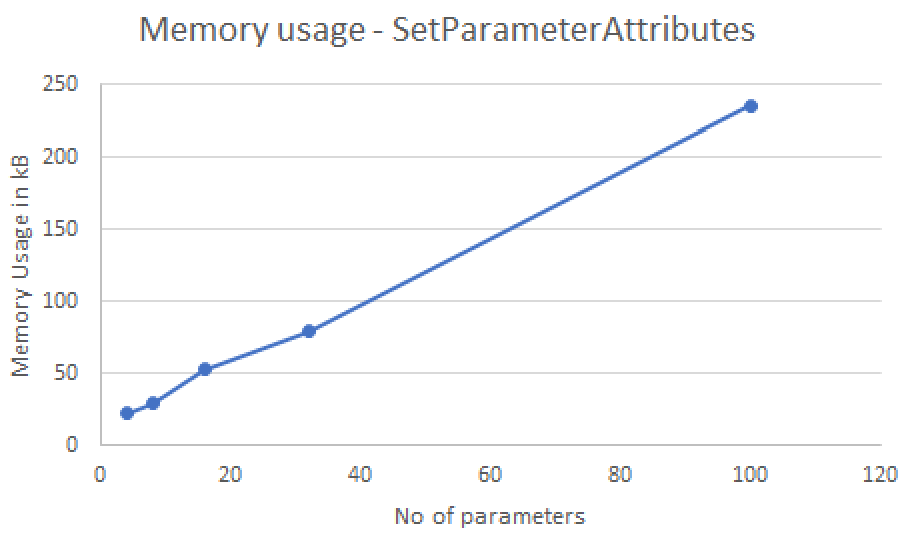


*Figure 12: SPA memory usage*

## 4.7. Análisis del consumo de memoria para GPA, GPN y GPV

Once identified GPA, GPN and GPV as the critical RPCs when the ACS requests the whole device datamodel (full tree), a deeper analysis was made in order to determine the more critical area among the two main tasks done in every RPC function:

- Glue Layer call: the retrieval of parameters is made by recursive calls that iterate over the datamodel.
- DOM Document: it is the DOM document building (the so called tree)
- based on the list returned by the Glue Layer.

The same test setup as described earlier was used to analyze these functions in deep. HSMEM_getMemUsage was employed not only to get the maximum memory usage as done in previous experiments but also to get the not freed memory. This is a key information as will show the amount of memory that will be still in use when the message building function (ixmlPrintDocument) is called.

| Function | Max usage (kB) | Not freed (kB) |
|---|---|---|
| **Glue Layer** | 51.2 | 19.5 |
| **DOM document building** | 182.9 | 173.2 |

*Table 6: GPA memory consumption*

| Function | Max usage (kB) | Not freed (kB) |
|---|---|---|
| **Glue Layer** | 51.73 | 17.1 |
| **DOM document building** | 109.3 | 99.8 |

*Table 7: GPN memory consumption*

| Function | Max usage (kB) | Not freed (kB) |
|---|---|---|
| **Glue Layer & DOM document building** | 148.4 | 117.7 |

*Table 8: GPV memory consumption*

## 4.8. Final conclusions and development proposals

The parameter retrieval performed by Glue Layer calls has a similar memory consumption between GPA and GPN. GPV was developed in a different way as previously presented in section X. Therefore, numbers cannot be splitted as done with the other two RPCs.

Only around a third of the 50kB needed by the Glue Layer is not freed once the parameter list is returned.

Nevertheless, DOM document build does an intensive memory usage, even three times bigger than Glue Layer call in GPA case. The most critical finding is that more than 90% of the memory allocated is not freed when the function returns. This is due to the fact that the whole datamodel is modelled in memory in the form of nodes representing a tree. This document will used later by the XML message build function and therefore can not be freed.

The following conclusions may be drawn:

- The biggest memory usage was caused by GPA, GPN and GPV when the ACS requests the whole datamodel. This RPCs are sent continuously during normal ACS operation mode and can compromise the stability of the system as total memory is almost exhausted.
- Getter and setter RPCs with a large number of parameters could need to allocate high amounts of memory. However, this is not a common behavior since predefined and custom RPCs usually include 3 or 4 parameters.
- GPA, GPN and GPV biggest consumptions are located in DOM document building task. Later, similar figures are observed to parse the document and build the response message.

Once the conclusions were showed to the MoCA team, based on the normal operation of the protocol and due to the temporal restrictions for the development, the final implementation decision was:

- Reduce the memory needed by GPA, GPN and GPV when the ACS requests the full device datamodel.

The proposal consisted on:

- Sending GPA, GPN and GPV large responses splitted in small messages. This will need the use of a standard mechanism understood by every ACS due to the fact that server code cannot be modified.
- Modify the behavior of each GPA, GPN and GPV to get individual parameters instead of returning a large DOM document. This way, we will save memory not only in this process but in the later one of parsing it and building the final XML message.

# 5. Development

The developing main purpose is to modify the CWMP client to slice the messages generated by RPCs GPA, GPN and GPV in small pieces. Therefore, it won’t be necessary to create a DOM representation of the requested tree and we will avoid the later parsing and subsequent XML response message build.

For that purpose, it is necessary to use a standard mechanism accepted by all ACS server as it is not possible to perform any modification in the server side.

## 5.1. Transfer-Encoding: Chunked

HTTP/1.1 allows the usage of ‘transfer-encoding’ headers applied to messages sent between a couple of node (CPE and ACS, in our case). ‘Chunked’ directive allows to split the data to send in a series of smaller fragments called ‘chunks’ [8].

‘Content-Length’ header showing information about the total size of the HTTP message is deleted and replaced by ‘Transfer-Encoding:chunked’. This header will tell the message receiver that the message is going to be splitter in several messages but final length and number of chunks are still unknown.

Fragmented messages are required to be built following some formatting rules: the length of the message to be sent in hexadecimal followed by a carriage return and a line feed. Next line will include the message and finally a carriage return and a line feed will indicate that the chunk is finished.

A message example using “Transfer-Encoding:chunked” directive is shown below:

```
66\r\n

<MaxEnvelopes>1</MaxEnvelopes><CurrentTime>2021-01-
26T12:16:31</CurrentTime><RetryCount>0</RetryCount>\r\n
```

This mechanism will allow us to send GPA, GPN and GPV by chunks avoiding the need of retrieving the full tree and knowing the number of parameters and the length of the final message beforehand.

The ‘Transfer-Encoding:chunked´ directive is a valid option since it is included in HTTP/1.1, a standard that must be implemented in all ACS servers. However, we decided to inspect the messages sent by CWMP protocol in a different product of the company called G.hn and verified this was the solution implemented to solve the same issue.

Figure 10 shows a Wireshark traffic capture of a G.hn inform message when the device boots up.

```
POST /ftacs-basic/ACS HTTP/1.1
Connection: Keep-Alive
Host: 95.216.70.142:8080
User-Agent: TR069 Client 1.0
Content-Type: text/xml; charset=utf-8
SOAPAction: ""
Transfer-Encoding: chunked

1a3
<?xml version='1.0' encoding='UTF-8'?><soap-env:Envelope xmlns:soap-env='http://
schemas.xmlsoap.org/soap/envelope/' xmlns:soap-enc='http://schemas.xmlsoap.org/soap/encoding/'
xmlns:xsd='http://www.w3.org/2001/XMLSchema' xmlns:xsi='http://www.w3.org/2001/XMLSchema-
instance' xmlns:cwmp='urn:dslforum-org:cwmp-1-2'><soap-env:Header><cwmp:ID soap-
env:mustUnderstand='1'>926516539</cwmp:ID></soap-env:Header><soap-env:Body>
d
<cwmp:Inform>
9f
<DeviceId><Manufacturer>MaxLinear</Manufacturer><OUI>00139D</OUI><ProductClass>MaxLinearGhn</
ProductClass><SerialNumber>00139D001EEB</SerialNumber></DeviceId>

30
<Event soap-enc:arrayType='cwmp:EventStruct[1]'>
51
<EventStruct><EventCode>1 BOOT</EventCode><CommandKey></CommandKey></EventStruct>
8
</Event>
66
<MaxEnvelopes>1</MaxEnvelopes><CurrentTime>2021-02-09T11:56:53</CurrentTime><RetryCount>0</
RetryCount>
41
<ParameterList soap-enc:arrayType='cwmp:ParameterValueStruct[9]'>
7d
<ParameterValueStruct><Name>Device.RootDataModelVersion</Name><Value
xsi:type='xsd:string'>2.4</Value></ParameterValueStruct>
83
<ParameterValueStruct><Name>Device.DeviceInfo.HardwareVersion</Name><Value
xsi:type='xsd:string'>1_0</Value></ParameterValueStruct>
b3
<ParameterValueStruct><Name>Device.DeviceInfo.SoftwareVersion</Name><Value
xsi:type='xsd:string'>dw920_v2_x-WorkssysEval-HN SPIRIT.v7_8_r619+8+1_cvs</Value></
ParameterValueStruct>
```

*Figure 13: traffic capture of G.hn inform sent in chunks*

## 5.2. HTTP client modifications

It's been necessary to adapt the functionality of the HTTP client in order to be capable of sending messages by chunks.

Figure 14 shows the activity diagram of the TR69 HTTP Client. It basically checks whether we are sending a chunked RPC (GPA, GPN, GPV) or not. Only on first iteration headers are added calling HTTPClientSendRequest. Following chunks are sent by calling HTTPClientWriteData directly.

**TR69_SoapSend:**

This function adds SOAP headers only on first iteration if we are sending chunks or in every call if we are not chunked sending. Finally calls TR69_HttpCSend.

**TR69_HttpCSend:**

This function sends the data to the HTTP server. It calls HTTPClientSendRequest when the RPC is not a chunked one or if it is the first chunk. Following chunks are sent by calling HTTPClientWriteData.

**HTTPClientSendRequest:**

This function adds HTTP headers and writes the message to the socket.

The functionality when the content-length is unknown has been added: "HTTP send chunk" flag is set to the session. Afterwards headings are added and HTTPClientWriteData is called.

**HTTPClientWriteData:**

This function builds the chunks as described in section "Transfer-Encoding: Chunked", calculating the chunk size and adding all necessary carriage returns and line feeds. Finally, it delivers the message to the socket.

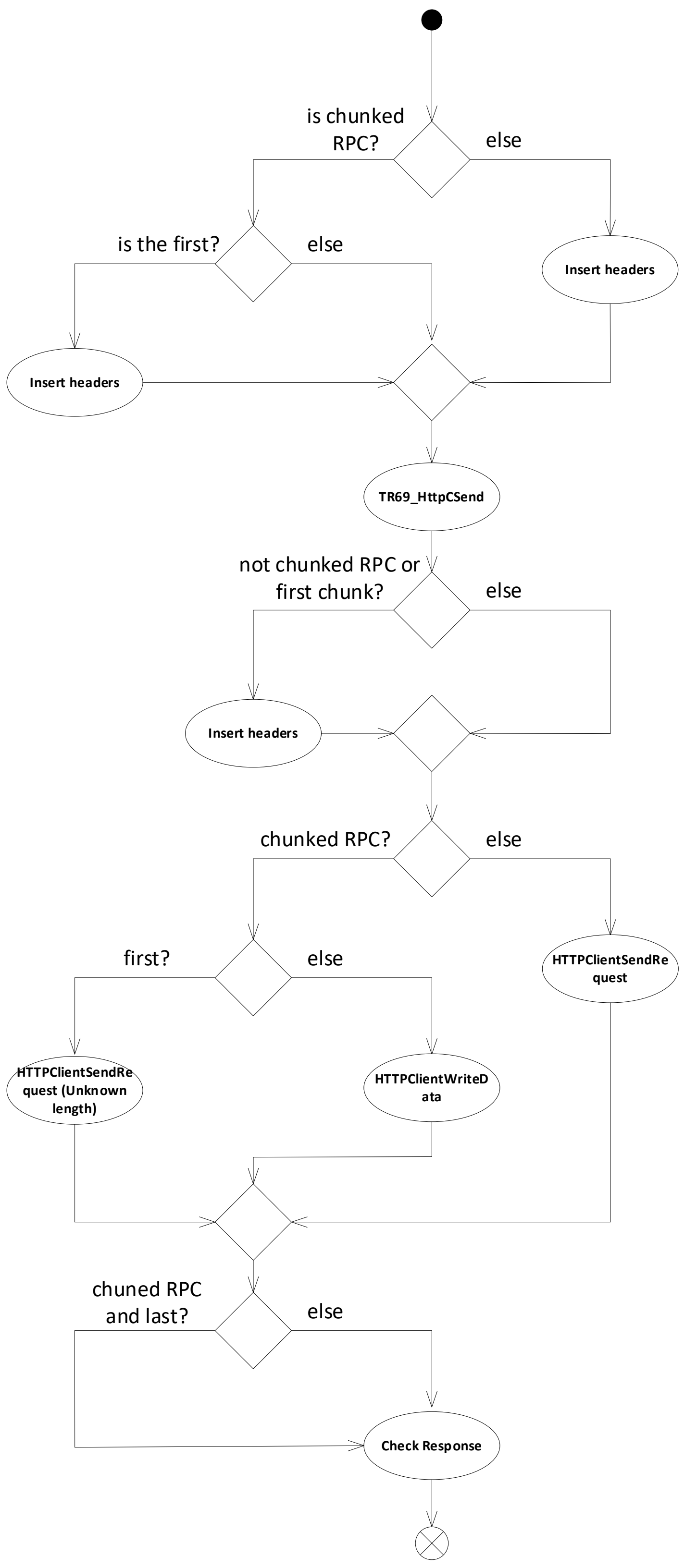


*Figure 14: HTTP client activity diagram*

## 5.3. CWMP client modifications

CWMP client modifications can be splitted in three big blocks described below:

- GPV glue layer call and DOM tree building must be decoupled: as presented in previous sections, this function differs from GPA and GPN in that the glue layer call does not only retrieve the parameters requested but also builds the DOM tree representation.
- Replace GPA, GPN and GPV functions with new implementations that do not build a DOM document.
- Implement new functions that build the chunk message to be sent each iteration, including headers and footers if needed.

Figure 15 shows the new CWMP sequence diagram:

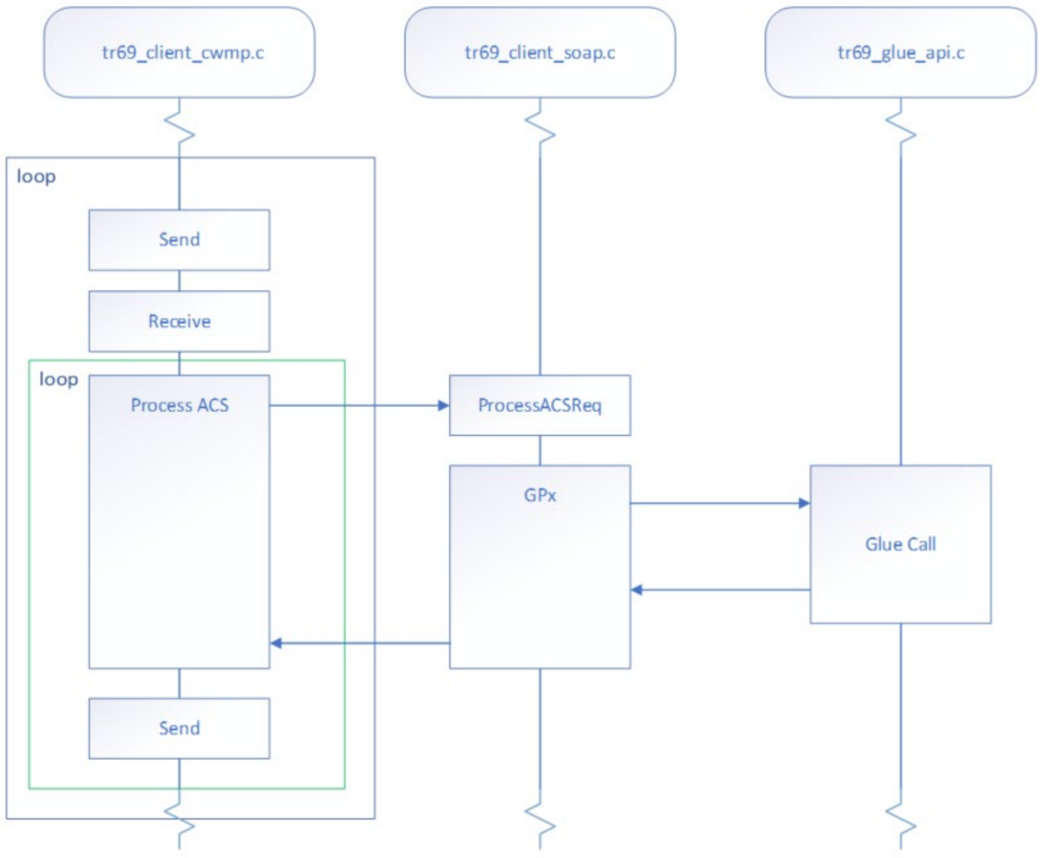


*Figure 15: new CWMP sequence diagram*

## 5.4. Implementation details

The new CWMP client implementation will iterate through the parameters retrieved by the Glue Layer in the main function of tr69_client_cwmp. This will differ from the previous implementation in which the iteration was done inside each RPC function. The new approach will avoid both task with the biggest consumptions: the DOM building and the parsing and message construction made by ixmlPrintDocument.

The Glue Layer will return all parameters requested only during the first iteration. New functions will be called to build the chunk message containing only the parameter corresponding with the iteration number.

### 5.4.1 Global Structure

A global structure is used to handle the new message sending by chunks. TR69 client execution is sequential and independent from other functionalities so there will be no data dependencies between functions.

```
typedef struct
{
  bool                      error;
  TR69_RPC_T                chunkedRpc;
  uint16_t                  numberNames;
  uint16_t                  loopCount;
  int                       msgStrIdx;
  int                       localBufferIdx;
  char                      *msgStr;
  char                      *localBuffer;
  char                      **gpnNameArray;
  char                      **gpaParamArray;
  bool                      *gpnWritableArray;
  TR69_PARAMATTR_T          **gpaAttrArray;
  TR69_PARAMVALUE_ERROR_T   **gpvParamArray;
 } TR69_CHUNK_SEND;
```

***error***

This boolean variable is responsible of tracking if an error is detected in the Glue Layer during the parameters retrieval. It could happen due to a incorrect path in the GPx function call, due to a wrong parameter name or any other issue found during the execution. If an error is detected, chunking sending is aborted. Instead a Fault response is sent in a traditional way.

***chunkedRpc***

It is an enum type that maintains the RPC that it is being processed:

```
typedef enum
{
        GET_RPC_METHODS,
        SET_PARAMETER_VALUES,
        GET_PARAMETER_VALUES,
        GET_PARAMETER_NAMES,
        SET_PARAMETER_ATTRIBUTES,
        GET_PARAMETER_ATTRIBUTES,
        ADD_OBJECT,
        DELETE_OBJECT,
        REBOOT,
        DOWNLOAD,
        FACTORY_RESET
} TR69_RPC_T
```

Only GET_PARAMETER_ATTRIBUTES, GET_PARAMETER_NAMES y GET_PARAMETER_VALUES support chunked sending.

***numberNames***

Stores the number of parameters returned by the Glue Layer. This value is obtained in the first GPx function call by ProcessACS.

It corresponds with the maximum number of iterations to perform and therefore with the number of messages to send.

***loopCount***

Number of iterations already performed (and messages already sent).

***msgStrIdx***

“msgStr” buffer index used to avoid buffer overflow when concatenating character strings. It is used by the functions that build the message to be sent.

***localBufferIdx***

“localBuffer” buffer index used to avoid buffer overflow when concatenating character strings. It is used by the functions that build the message to be sent.

***gpnNameArray***

Array returned by the specific Glue Layer function called by GPN. It is a linked list that contains the requested parameters.

***gpnWritableArray***

Array returned by the specific Glue Layer function called by GPN. It is a linked list that contains the attributes of each requested parameter.

***gpaAttrArray***

Array returned by the specific Glue Layer function called by GPA. It is a linked list that contains the attributes of each requested parameter.

***gpvParamArray***

Array returned by the specific Glue Layer function called by GPV. It is a linked list that contains the requested parameters.

### 5.4.2. GPA, GPN and GPV functions

TR69_rpcGetParameterAttributes, TR69_rpcGetParameterNames and TR69_rpcGetParameterValues have been modified in the following way: only on first iteration, when “loopCount” is ‘0’, they call their corresponding Glue Layer function. This returns a linked list stored by reference in the global structure showed before. The total number of parameters that the list includes is stored in “numberNames”.

This way, the total number of iterations and therefore the messages to be sent is known.

If an error is detected during the Glue Layer call, error global variable will be set to ‘1’. A Fault message will be built and chunk sending will be aborted.

If the Glue Layer retrieves successfully the requested parameters, new functions TR69_xmlGetParameter[Attributes, Names, Values] will be called to build the chunk corresponding to the current iteration.

### 5.4.3. XML message build functions

New functions called TR69_xmlGetParameterAttributes, TR69_xmlGetParameterNames and TR69_xmlGetParameterValues are responsible of building the chunk string to be sent in each iteration. They are called by each GPx function.

The string is stored in “msgStr” buffer while “localBuffer” is used to retrieve every parameter to be appended in “msgStr”. As C language does not provide automatic mechanism to avoid buffer overflow, “msgStrIdx” and “localBufferIdx” presented in previous section are used not to exceed the maximum length for these buffers.

Only on first iteration and last iteration appropriate headers and footers will be included in the message.

If an overflow occurs while building the chunk (whether the parameter does not fit in “localBuffer” or the string to be sent does not fit in “msgStr”), an error status is returned.

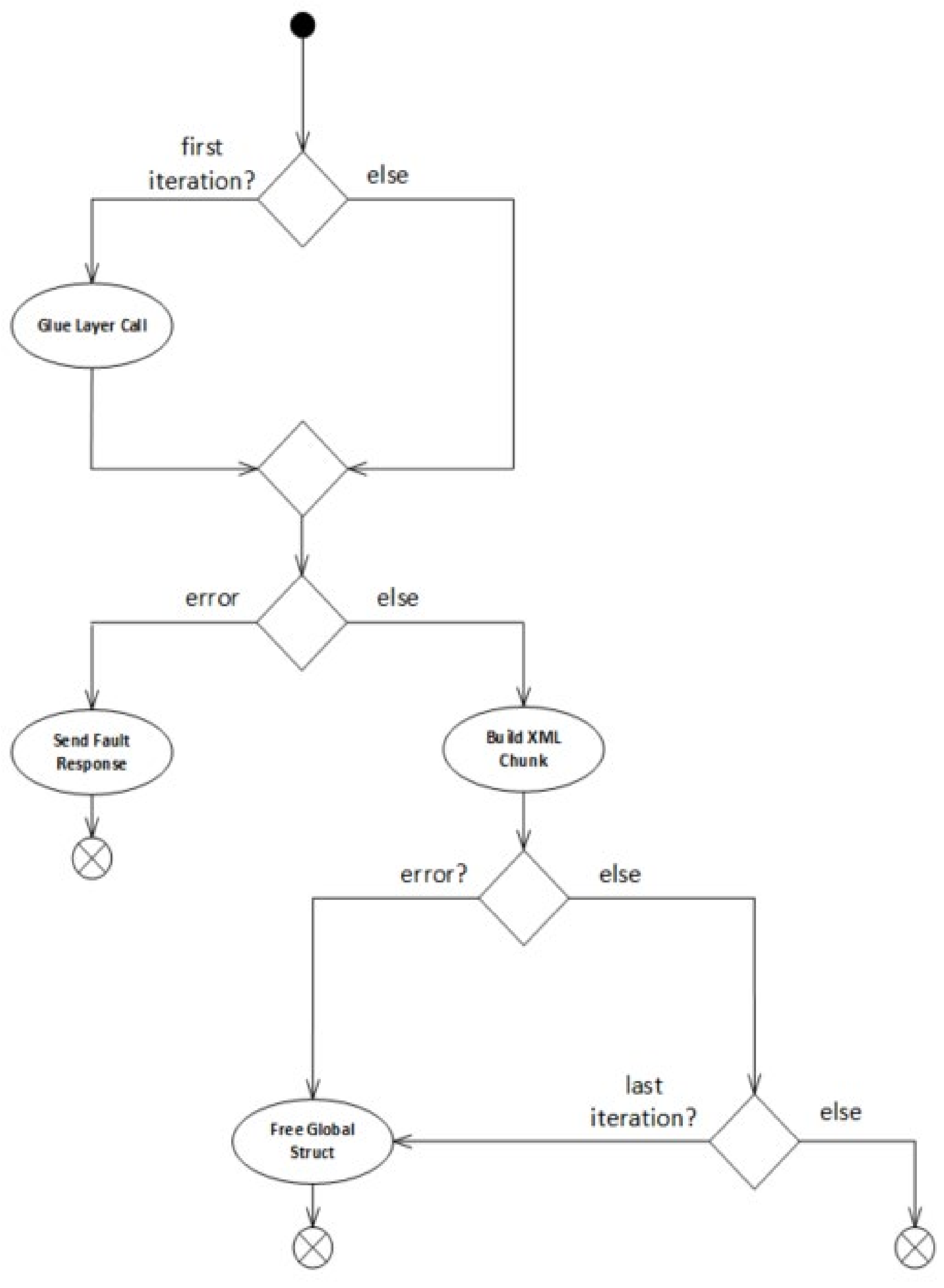


*Figure 16: GPA, GPN y GPV activity diagram*

### 5.4.4. Main function

Main CWMP client function has been partially modified: HTTP session creation, inform sending, inform-response reception and firmware download process have not been reimplemented. Therefore, Figure 14 activity diagram only shows the RPC management code block.

The new implementation is in charge of getting the number of iterations to be performed. This number will be retrieved by the Glue Layer and stored in “numberNames” global variable as shown in before.

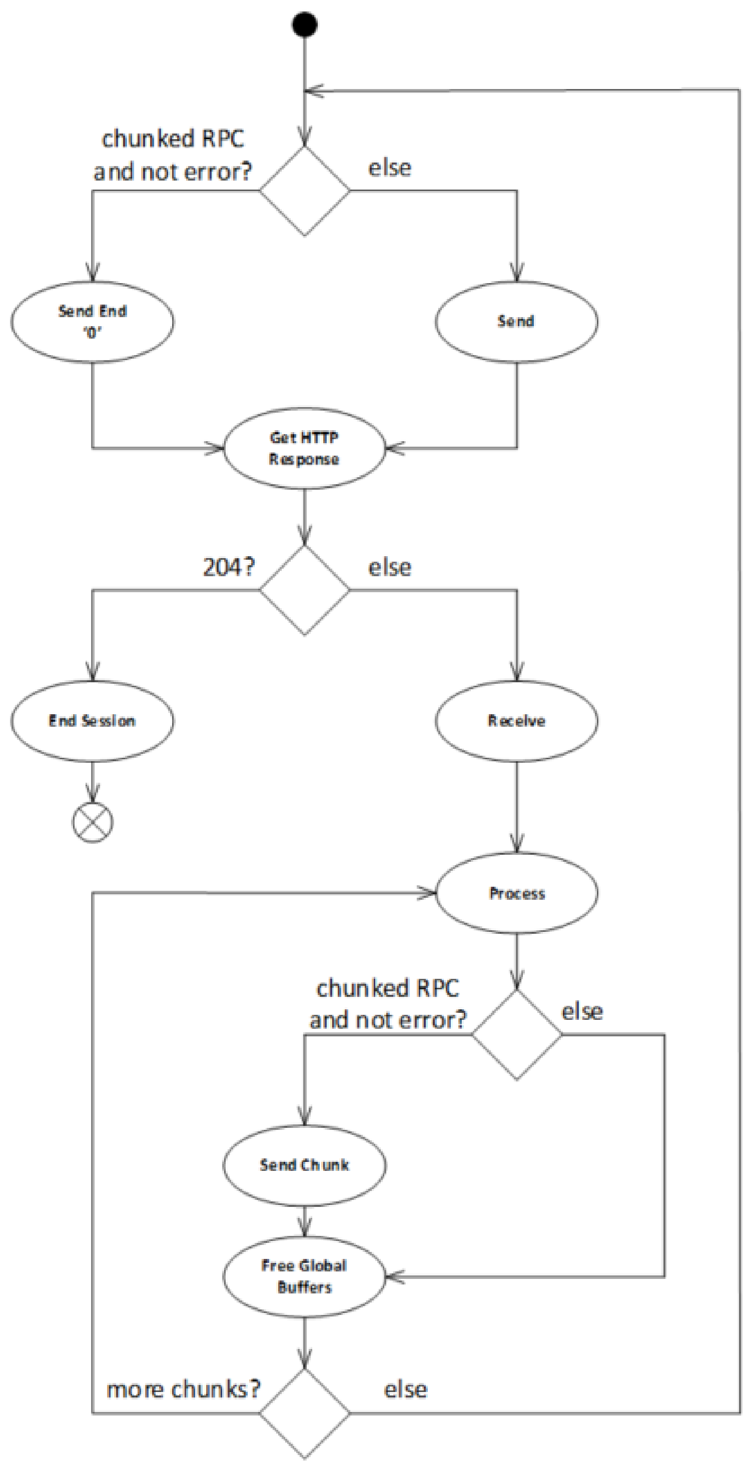


*Figure 17: TR-069 main function activity diagram*

# 6. Final memory layout

Once the development was concluded, next step was to measure the new memory needs to compare with the previous implementation and define a new memory layout that optimizes the memory block adjusted to the new consumptions.

First measurements were done using the MoCA simulator used during the whole development process. HSMEM_getMemUsage and FriendlyACS were used the same way as presented in chapter 4 to retrieve the information needed.

After that, the new solution was tested using a unique modem and once checked that the firmware was correct and the device did not stop working, several tests were made in a TR-069 test bench.

## 6.1. Consumption measurements

GPA, GPN and GPV requesting the whole datamodel were sent to a couple of simulator instances: the first one was loaded with the current firmware while the second one implemented the chunk sending.

The memory usage represents the maximum memory needed by the CWMP client to perform each a full connection receiving each RPC.

| RPC | Number of parameters | Memory usage (kB) Current implementation | Memory usage (kB) New implementation | Memory Savings |
|---|---|---|---|---|
| **GPA** | Full tree | 424.95 | 50.21 | 88.16% |
| **GPN** | Full tree | 346.5 | 50.67 | 85.35% |
| **GPV** | Full tree | 374.5 | 53.66 | 85.67% |

*Table 9: GPA, GPN and GPV memory usage comparison*

Table 9 results showed a reduction of more then 85% in all of the three RPCs.

Next step was to determine how this drastically memory reduction affected to the block usage. As presented in 4.1. Current memory layout, memory was allocated at the device boot following a block strategy showed in Table 1. Memory layout needs to be created ad-hoc so we can optimize allocations.

While simulation product was a valuable tool for our developing purposes, determining the new memory layout is hardware specific as real modem implementation differs from simulator in the IP stack used among many other software parts.

For the new memory layout definition, a specific TR-069 bench with real modems was used. Friendly ACS was also used but time modems are isolated and cannot connect to the internet so a local version of the ACS was employed.

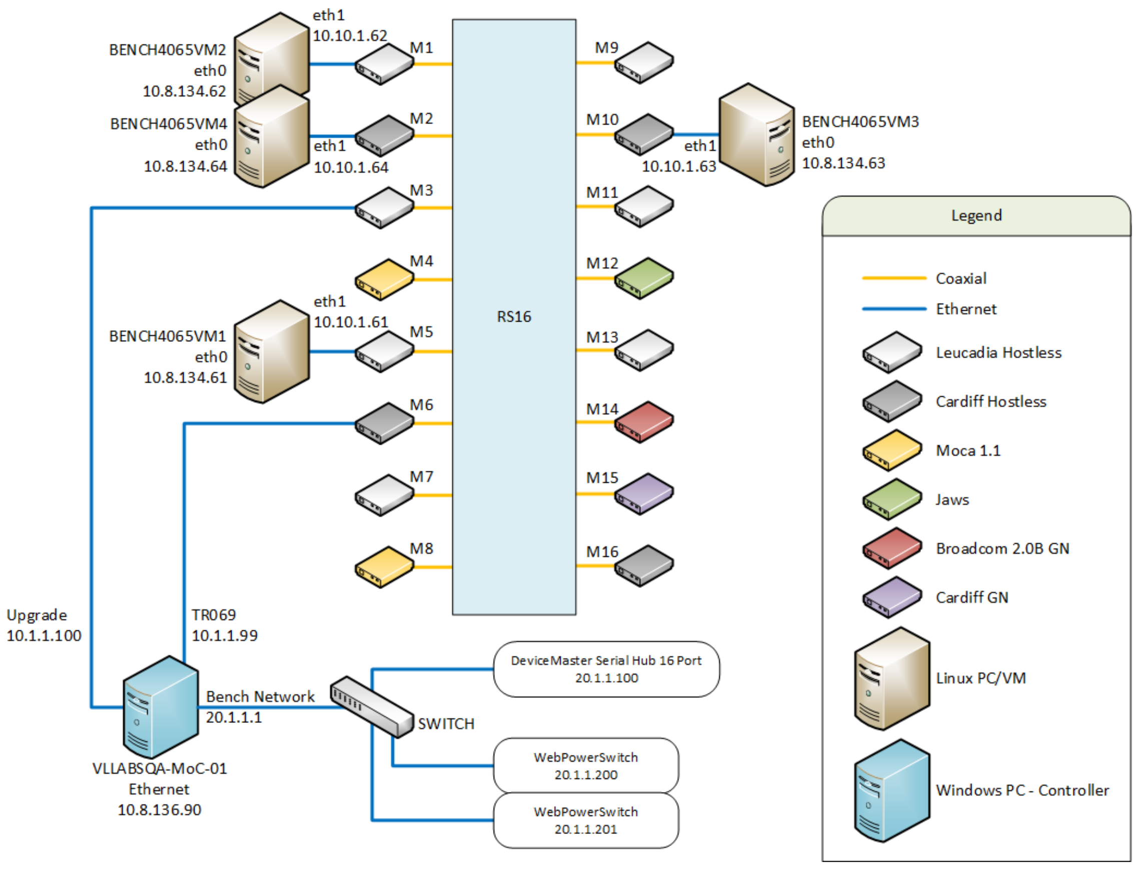


*Figure 18: TR-069 testbench setup*

The experiment consisted on the CPE registration into the ACS, a normal behavior that usually involves the use of many RPCs. These are:

- Inform message.
- Full datamodel GPA, GPN y GPV.
- 4 parameters SPV.

Table 10 shows that many blocks are already at their maximum capacity. Many others are near 100%. This is the expected behavior as this memory layout was intentionally created to optimize the block usage for the current CWMP implementation.

| Block Size (kB) | Number of blocks | Max. used blocks | Use % |
|---|---:|---:|---:|
| **16** | 2300 | 2300 | 100 |
| **32** | 1250 | 1067 | 85 |
| **72** | 1900 | 1736 | 91 |
| **256** | 20 | 15 | 75 |
| **360** | 10 | 9 | 90 |
| **1024** | 15 | 7 | 57 |
| **2048** | 10 | 4 | 40 |
| **3072** | 4 | 3 | 75 |
| **4096** | 3 | 2 | 67 |
| **13312** | 2 | 2 | 100 |
| **18432** | 2 | 2 | 100 |
| **59392** | 2 | 2 | 100 |

*Table 10: current memory block usage*

The experiment was repeated this time with the new firmware versión chunk sending. A drastical reduction is shown in Table 11 where we can find that any of the blocks is at its 100% capacity. Even some of them are no longer used.

| Block Size (kB) | Number of blocks | Max. used blocks | Use % |
|---|---:|---:|---:|
| **16** | 2300 | 482 | 21 |
| **32** | 1250 | 322 | 26 |
| **72** | 1900 | 453 | 24 |
| **256** | 20 | 15 | 75 |
| **360** | 10 | 9 | 90 |
| **1024** | 15 | 7 | 47 |
| **2048** | 10 | 4 | 40 |
| **3072** | 4 | 2 | 50 |

| | | | |
|---|---:|---:|---:|
| **4096** | 3 | 1 | 33 |
| **13312** | 2 | 0 | 0 |
| **18432** | 2 | 2 | 100 |
| **59392** | 2 | 0 | 0 |

*Table 11: new memory block usage*

Biggest blocks are no longer used with the exception of the 18kB blocks which are used for TLS library as described in some comments found in the code.

With this new scenario, the failure probability is less than before: if an unexpected allocation of memory takes place or even the datamodel grows, the system will have enough memory not to fail. However, keeping this memory layout leads us to waste memory as all these blocks are reserved at the boot time for the CWMP client.

## 6.2. New memory layout

The proposal for the new memory layout deletes no longer used blocks and reduces the rest of them so that approximately a 30% of spare memory is kept at every block size. This is done to reduce the possibility of having an allocation failure due to an unexpected memory allocation. 18kB blocks are not modified since these are used by TLS input/output buffer as previously discussed.

| **Block Size (kB)** | **Number of blocks** | **Max. used blocks** | **Use %** |
|---|---:|---:|---:|
| **16** | 730 | 482 | 66 |
| **32** | 520 | 322 | 62 |
| **72** | 650 | 453 | 70 |
| **256** | 25 | 15 | 60 |
| **360** | 13 | 9 | 69 |
| **1024** | 10 | 7 | 70 |
| **2048** | 6 | 4 | 67 |
| **3072** | 6 | 2 | 33 |
| **4096** | 2 | 1 | 50 |
| **18432** | 2 | 2 | 100 |

*Table 12: new memory layout and usage*

# 7. Future improvements

The improvements presented in this project are focused on reducing the memory usage by the most common RPC: GPA, GPN and GPV of full tree. However, as discusses previously, getter and setter calls with a large number of parameters, although not usually sent, can become an issue: CWMP client will need a big buffer to allocate the received message and DOM document building will be built for SPx remote procedure calls.

Since auto configuration servers do not split messages in chunks, the CWMP client could be modified in order to process the received message by events instead of building a DOM document. This change will need to replace the DOM parser by a SAX one.

Although Inform sending, Fault error responses and the rest of the RPCs do not need big amounts of memory to perform their work, sending them in chunks will bring homogeneity to the client.

Being aware of the limitations and knowing which are the potential improvements allow to resume these tasks in a future if needed.

# 8. Final conclusions

At the beginning of this project, MoCA modems using CWMP were almost at the limit of their memory capacity due to the big consumption the protocol needed to perform its work. This high memory usage limited the possibility of adding new features to the product and compromise the modem performance as packet transmission buffers were reduced to fit the remote management protocol. Moreover, some ACS send several requests at the same time what caused device failures and constant reboots.

The main goal of this project was to reduce the memory usage needed for CWMP operation. For this purpose, it was necessary to perform a deep analysis that help us to determine which were the most conflictive components of TR-069 implementation.

A complementary objective was to run the MoCA product simulator with the aim of helping not only during the development of this project but with future ones.

Simulator fixes and findings allow us to set up a local testing environment. Using the simulator helped to avoid some of the issues that a developer faces when developing embedded software and eased the measurements.

Experiments thrown that GPA, GPN and GPV remote procedure calls were the most memory demanding functions when the whole CPE datamodel was requested. The main issue was related with the need of building a DOM document containing all the parameters requested and lately processing it to build the final XML message.

The proposal developed in this project consisted on sending GPA, GPN and GPV response messages splitted in many messages as parameters were retrieved. For this purpose, a fragmented sending mechanism supported by HTTP/1.1 was used: ´Transfer-Encoding:chunked’.

It has been necessary to modify the CWMP client so the Glue Layer, in charge of parameter retrieval, was only called during the first iteration: the loop that went through the parameter list building the DOM document has been moved to the CWMP main function. This way, it is no longer necessary to build the DOM representation of the datamodel in memory. Instead, new message building functions have been developed in order to construct a lightweight chunk message.

It was also necessary to adapt the HTTP client in order to allow chunk sending. As discussed before, these messages does not follow the standard structure of an HTTP message.

Once the code was tested in the simulation environment and checked that worked in real modems, it was necessary to determine the memory consumption reduction. For that purpose, a dedicated TR-069 test bench was used as memory needs are different for simulator and real products.

The new memory layout reduces the number of blocks and even remove some of them. This new schema uses 188.272kB instead of the 487.080kB needed by the previous implementation. A total of 298.808kB have been freed and could be used to increase MoCA transmission buffers or develop new features. This reduction represents a 7.5% of the total 4MB system memory.

# 9. Bibliography

[1] G. Davis. (2018). 2020: Life with 50 billion connected devices. *2018 IEEE International Conference on Consumer Electronics (ICCE)*, 2018, pp. 1-1, doi: 10.1109/ICCE.2018.8326056.

[2] S. Kemp. (2021, January 27th). Digital 2021: Global Overview Report. Datareportal. https://datareportal.com/reports/digital-2021-global-overview-report

[3] TR-069 CPE WAN Management Protocol. (2020, June). Broadband Forum. https://www.broadband-forum.org/technical/download/TR-069_Amendment-6_Corrigendum-1.pdf

[4] Box et al. (2000). Simple Object Access Protocol (SOAP) 1.1. The World Web Concostium, W3C. https://www.w3.org/TR/2000/NOTE-SOAP-20000508/

[5] I. Savic, M.S. Savic, G. Velickic. (2014). Implementation of TR-069 connection request mechanism. X International Symposium on Industrial Electronics INDEL, Banja Luka.
http://indel.etfbl.net/2014/resources/Proceedings_2014/INDEL_2014_Paper_44.pdf

[6] DOM Living Standard. Web Hypertext Application Technology Working Group. https://dom.spec.whatwg.org/

[7] C. Yao and Q. Li. (2003). 13.3. Fixed-Size Memory Management in Embedded Systems. *Real-Time Concepts for Embedded Systems*. CRC Press.

[8] R. Fielding and J. Reschke. (2014). Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing. RFC 7230, DOI 10.17487/RFC7230. https://datatracker.ietf.org/doc/html/rfc7230