

POLITÉCNICO DO PORTO
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

Estudo/Implementação de uma solução 4G

José Nuno Loureiro Novais da Silva

LEEC
Licenciatura em Engenharia Eletrotécnica e de Computadores



DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA
Instituto Superior de Engenharia do Porto

Junho, 2021

Este relatório satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Projeto/Estágio, do 3º ano, da Licenciatura em Engenharia Eletrotécnica e de Computadores.

Candidato: José Nuno Loureiro Novais da Silva, N.º 1180896,
1180896@isep.ipp.pt

Orientação Científica: Isabel Maria de Sousa de Jesus, isj@isep.ipp.pt

Empresa: DMS, Displays & Mobility Solutions

Orientador: Francisco Pérola, franciscoperola@dmsdisplays.com



DEPARTAMENTO DE ENGENHARIA ELETROTÉCNICA
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200-072 Porto

Junho, 2021

Agradecimentos

Nesta secção, quero agradecer a todas as pessoas que contribuíram para o sucesso do meu percurso académico, nomeadamente à minha família, com um agradecimento especial à minha namorada. Quero agradecer também aos meus amigos e aos meus pais e irmãos que, desde o início da minha caminhada académica, ajudaram e empurraram para cumprir os meus objetivos.

Mais especificamente, quero agradecer à Eng. Isabel Jesus, pelo acompanhamento prestado e a dedicação, tanto nas aulas que lecionou no meu primeiro e segundo ano, como na elaboração deste relatório.

Finalmente, um agradecimento à DMS, por me ter acolhido e também ao meu orientador, Francisco Pérola, pelo acompanhamento incansável ao longo do projeto.

Resumo

No âmbito da Unidade Curricular de Estágio, do 3º ano da Licenciatura em Engenharia Eletrotécnica e de Computadores irá ser desenvolvido um projeto que permitirá estabelecer a ligação entre os autocarros de uma área metropolitana, um servidor e as paragens dos mesmos.

Através da implementação desta solução, será possível exibir com precisão os intervalos de tempo até passar o próximo transporte em uma determinada estação, facilitando e oferecendo uma experiência mais eficiente e fiável aos vários utilizadores diários que recorrem a este serviço para se deslocar.

O principal objetivo deste projeto é realizar um estudo sobre as diferentes tecnologias disponíveis, neste caso CAT 1 e Narrowband-IOT (NB-IOT), determinar o hardware necessário para implementar a solução desenvolvida e estabelecer protocolos de comunicação e troca de informação entre servidor e modem.

Para a concretização destes objetivos estudou-se o modem CAT 1, estabeleceram-se através do protocolo de comunicação TCP ligações modem/servidor, e foram retiradas as latências de comunicação. O objetivo seguinte foi medir e registar o consumo energético do modem.

Após ter os dados necessários relativamente ao modem CAT 1, efetuou-se o mesmo procedimento para o modem NB-IOT, mediram-se as latências e consumos, em transferência de dados, idle e em Power Save Mode(PSM).

Avaliando os consumos energéticos e latências de cada modem e comparando-os, apesar do modem CAT 1 apresentar consumos mais elevados, optou-se pela sua utilização devido às latências de comunicação serem mais baixas do que as do modem NB-IOT. Na última fase do projeto foi desenvolvida uma aplicação em linguagem C a funcionar em ambiente Linux, com o objetivo de controlar o modem.

Palavras-Chave: Modem, CAT1, NB-IOT, Power Save Mode, API, Linux

Abstract

Within the scope of the Internship Curricular Unit, of the 3rd year of the Degree in Electrical and Computer Engineering, a project will be developed that will allow the establishment of a connection between buses in a metropolitan area, a server and their stops.

By implementing this solution, it will be possible to accurately display the time intervals until the next transport passes at a provided station, facilitating and offering a more efficient and reliable experience for the various users who use this service to get around.

The main objective of this project is to carry out a study on the different technologies available, in this case CAT 1 and NarrowBand-IOT (NB-IOT), determine the necessary hardware to implement the developed solution and establish communication protocols and information exchange between server and modem.

To achieve these objectives, the CAT 1 modem was studied, modem/server connections were established through the TCP communication protocol, and communication latencies were collected. The next objective was to measure and record the modem's energy consumption.

After having the necessary data regarding the CAT 1 modem, the same procedure was carried out for the NB-IOT modem, measuring latencies and consumption, in data transfer, idle and in Power Save Mode (PSM).

Assessing the energy consumption and latencies of each modem and comparing them, although the CAT 1 modem has higher consumption, we chose to use it because the communication latencies are lower than those of the NB-IOT modem.

In the last phase of the project, an application in C language was developed to work in a Linux environment, with the objective of controlling the modem.

Keywords: Modem, CAT1, NB-IOT, Power Save Mode, API, Linux

Índice

Lista de Figuras	vii
Lista de Tabelas	ix
Listagens	xi
Lista de Acrónimos	xiii
1 Introdução	1
1.1 Contextualização	2
1.2 Objetivos	2
1.2.1 Fases do Projeto	2
1.3 Calendarização	3
1.4 Organização do Relatório	3
2 Hardware	5
2.1 Modem	5
2.2 Modem CAT1-LE910C1-EU	6
2.3 Modem NB-IOT - BC66	8
2.4 Raspberry Pi	9
3 Desenvolvimento do Projeto	11
3.1 Descodificação Mensagens	11
3.2 Ligação TCP	13
3.2.1 Servidor	13
3.2.2 One Time Setup	15
3.2.3 Initial Setup	16
3.2.4 TCP	16
3.3 Power Save Mode	17
3.4 Recolher Latências e Consumos - CAT1	18
3.4.1 Scripts - Docklight	18
3.4.2 Latências - CAT1	20
3.4.3 Consumos - CAT1	21
3.5 NBIOT	23

3.5.1	Latências - NB-IOT	23
3.5.2	Consumos - NB-IOT	23
	PSM e eDRX	23
	Exemplo Cálculo T3324	24
3.6	Resultados Obtidos	26
3.6.1	Latências CAT1 vs NB-IOT	26
3.6.2	Consumos CAT1 vs NB-IOT	26
4	Implementação da API	27
4.1	Função ligação ao modem	27
4.2	Função enviar dados	29
4.3	Função ler dados	29
4.4	Função escrever comandos	29
4.5	Função Ler/Escrever	30
4.6	Verificar PSM	31
4.6.1	Set DTR	32
4.6.2	Clear DTR	32
4.7	One Time Setup	33
4.8	Initial Setup	33
4.9	Ligação TCP	34
4.10	Fechar ligação TCP	34
4.11	Entrar em PSM	35
4.12	Sair PSM	35
4.13	Menu de opções	36
4.14	Main	37
4.15	Macros	38
4.16	Exemplos de Resultados Obtidos	40
5	Conclusões	43
	Referências	44
A	Código Integral	49
A.1	Main.c	49
A.2	Main.h	61

Lista de Figuras

1.1	Calendarização do Projeto	3
2.1	Rede Modem[1]	5
2.2	Aplicações do NB-IOT, CAT1, CAT4[2]	6
2.3	LE910C1-EU c/ evaluation board	7
2.4	Diagrama de blocos do modem CAT1[3]	7
2.5	Quectel BC66[4]	8
2.6	Raspberry Pi	9
2.7	Raspberry Pi c/ alimentação	9
3.1	Teste comandos AT	12
3.2	Mensagem Recebida	12
3.3	Serial Port Monitor	13
3.4	Regra Firewall(1)	13
3.5	Regra Firewall(2)	14
3.6	Configuração Porta	14
3.7	Servidor TCP - Hercules	15
3.8	Ligação TCP	17
3.9	Diagrama de blocos PSM	17
3.10	Modo PSM	18
3.11	Modem Docklight Cliente	19
3.12	Docklight Servidor	19
3.13	Jumpers PL105/PL106	21
3.14	Jumpers PL109/PL110	21
3.15	Esquema Alimentação	22
3.16	T3412 e T3324[5]	24
3.17	PSM e eDRX[6]	25
4.1	Menu API	40
4.2	Initial Setup API	40
4.3	Ligação TCP API	41
4.4	Enter PSM API	41
4.5	Fechar ligação TCP API	42
4.6	Desligar API	42

A.1 Fluxograma base API	63
-----------------------------------	----

Lista de Tabelas

3.1	Formato de Dados	11
3.2	Latência CAT1	20
3.3	Consumo CAT1	22
3.4	Latência NB-IOT	23

Listagens

4.1	Abrir Porta Série	28
4.2	Escrever Dados	29
4.3	Ler Dados	29
4.4	Escrever Comandos AT	30
4.5	Ler e Escrever	30
4.6	Check CFUN	31
4.7	Set DTR	32
4.8	Clear DTR	32
4.9	One Time Setup	33
4.10	Initial Setup	33
4.11	Abrir ligação TCP	34
4.12	Fechar ligação TCP	34
4.13	Entrar em PSM	35
4.14	Sair PSM	35
4.15	Função Menu	36
4.16	Main	37
4.17	ficheiro Header	38
A.1	Main	49
A.2	Header	61

Lista de Acrónimos

3GPP	3rd Generation Partnership Project
API	<i>Application Programming Interface</i>
ASCII	American Standard Code for Information Interchange
AT Commands	Attention commands
CR	Carriage Return
DEE	Departamento de Engenharia Electrotécnica
ISEP	Instituto Superior de Engenharia do Porto
ISP	Internet service provider
LEEC	Licenciatura em Engenharia Eletrotécnica e de Computadores
LPWAN	low power wide area network
NB-IOT	Narrowband-IOT
PSM	Power Save Mode
TCP	Transmission Control Protocol

Capítulo 1

Introdução

O presente relatório de estágio expõe todos os conteúdos referentes ao projeto desenvolvido na Display & Mobility Solutions, realizado no âmbito da unidade curricular de Projeto/Estágio do 3º ano da Licenciatura em Engenharia Eletrotécnica e de Computadores (LEEC), do Departamento de Engenharia Electrotécnica (DEE), do Instituto Superior de Engenharia do Porto (ISEP).

No início de 2015 foi criada a DMS - Displays & Mobility Solutions, Lda. Com o objetivo de focar e desenvolver a sua atividade em sistemas de informação pública e para a indústria dos transportes.

A DMS é um fornecedor de soluções para produtos e sistemas eletrónicos para informação pública e soluções de segurança e mobilidade para a indústria dos transportes.

A DMS projeta e desenvolve equipamentos de informação pública, ou seja, painéis eletrónicos de exibição, tal como:

- Sinais de mensagens variáveis para coordenação de tráfego e segurança.
- Exibições de informações na plataforma para transportes públicos.
- Placas de estacionamento eletrónicas para informações e administração de estacionamento.
- *Displays* eletrónicos para diferentes tipos de aplicações.

1.1 Contextualização

O conceito deste projeto resultou da necessidade, de atualizar o método de disponibilização das informações referentes aos transportes que estão em vigor.

A necessidade de se alterar a utilização dos horários em papel, advém do facto de que o papel é apenas uma referência para o utilizador, pois não se consegue prever as irregularidades e discrepâncias dos horários dos transportes públicos devido a acidentes ou trânsito.

Assim, foi proposto criar uma solução *low power* e eficiente, que através de um servidor estabelecesse contacto entre os transportes e a informação existente nos painéis informativos, atualizando ao minuto o estado e intervalo de tempo até ao próximo transporte.

1.2 Objetivos

O objetivo deste trabalho é a implementação de uma *Application Programming Interface (API)*, desenvolvida em linguagem C, a correr em ambiente *linux*, dedicada não só a receber e enviar dados através de um modem, mas também a processá-los. Nas subsecções seguintes é apresentado as fases do projeto, a sua calendarização representada pelo Diagrama de Gantt e a organização do relatório nos seus diversos capítulos.

1.2.1 Fases do Projeto

- Ler datasheets/ aprender a utilizar o modem CAT1 através da aplicação AT Controller
- Decodificação de mensagens - *serial port monitor*
- Estabelecer comunicação Transmission Control Protocol (TCP) e *Power Save Mode (PSM)*
- Criação de *scripts* para facilitar a comunicação – *docklight*
- Recolher as latências e consumos do modem CAT1
- Estudo do modem NB-IOT
- Através dos *scripts* já criados, adaptar e fazer os mesmos testes para o modem NB-IOT
- Verificar qual dos modems é mais eficiente e fazer a escolha de qual utilizar
- Desenvolver API
- Conclusões

1.3 Calendarização

De modo a cumprir os prazos estabelecidos, o desenvolvimento do projeto foi calendarizado, de acordo com a cronologia apresentada na Figura 1.1.



Figura 1.1: Calendarização do Projeto

O projeto iniciou-se com um estudo do modem CAT1, aprender sobre os *at commands*, descobrir como os dados são enviados do computador para o modem, através do *datasheet* posicionar os *jumpers* nos pinos corretos para alimentar com uma fonte externa apenas o módulo/processador do modem e poder medir o consumo de energia. Esta fase do projeto durou 2 semanas, com início a 1 de março até 15 de março.

Numa fase seguinte testou-se as comunicações TCP/UDP e os vários modos de energia do modem, teve duração de 5 semanas.

Após ter estabelecido os modelos para a criação de ligações TCP e manter o modem em modo PSM, procedeu-se à criação de scripts para agilizar o processo de trabalhar com o modem. Esta fase durou sensivelmente 2 semanas.

Repetiu-se o processo de estudo e teste no modem NB-IOT, a experiência de ter trabalhado já com o modem CAT1 refletiu-se e apenas demorou 2 semanas.

A parte final do projeto foi o desenvolvimento da API, que demorou 3 semanas a ser concluída.

Por fim, a última fase é dedicada à escrita do relatório.

1.4 Organização do Relatório

No Capítulo 1 é feita uma apresentação à empresa onde este estágio foi realizado, introduzidos os objetivos deste projeto, a calendarização do trabalho e descrita a organização neste documento.

No Capítulo seguinte, 2, é feita uma introdução ao hardware que foi escolhido pela empresa para ser testado e desenvolvido o projeto.

No Capítulo 3 é desenvolvido os vários passos de estudo, desenvolvimento e implementação do projeto.

Capítulo 2

Hardware

2.1 Modem

O modem é o dispositivo eletrônico que estabelece a conexão entre o nosso computador e a internet[7]. Converte os sinais digitais enviados pelo computador em analógicos e reencaminha-os para o *Internet service provider (ISP)*, quando recebemos dados, o modem converte os sinais analógicos em digitais, representado na Figura 2.1.

A própria palavra é uma forma abreviada de Modulador-Demodulador, como o dispositivo executa a modulação e demodulação de sinais analógicos a sinais digitais [8].



Figura 2.1: Rede Modem[1]

2.2 Modem CAT1-LE910C1-EU

Cat1 é uma tecnologia antiga (release 8 do 3rd Generation Partnership Project (3GPP)), contudo amadurecida e bastante implementada e abrangida pela maioria dos ISP. Designada para dispositivos IOT com necessidade de largura de banda, baixa e média.

A tecnologia funciona no sistema 4G e se necessário nos 3G e 2G. Apresenta velocidades de download na ordem dos 10 Mbps, 5 Mbps de upload e latências entre os 50 e 100 ms[9].

O modem CAT1 é bastante flexível, devido ao facto de conseguir trabalhar com aplicações de baixo consumo, tal como o NB-IOT, mas se necessário suporta alta largura de banda (20 Mhz).

Esta tecnologia é utilizada atualmente em caixas de multibanco, vídeo vigilância, drones autónomos, *smart cities*, etc[10]. Apesar do suporte de altas larguras de banda os modems CAT1 não são suficientes para suportar as necessidades de veículos autónomos ou aplicações de vídeo em tempo real que necessitam de larguras de banda maiores, como se pode ver na Figura 2.2.

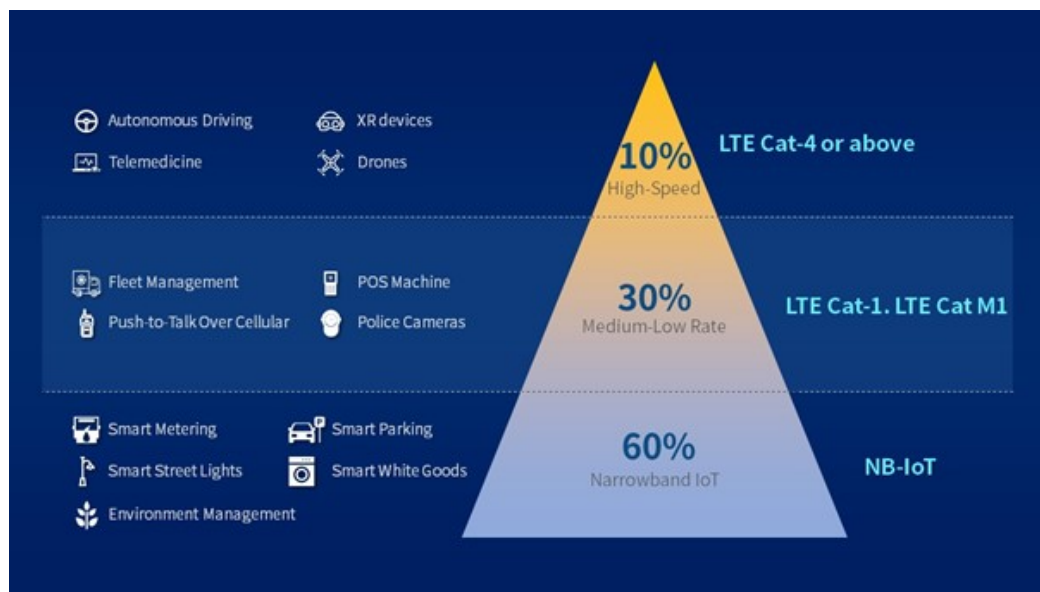


Figura 2.2: Aplicações do NB-IOT, CAT1, CAT4[2]

O modem seleccionado pela DMS foi o LE910C1-EU, Figura 2.3 com o seguinte diagrama de blocos, Figura 2.4.



Figura 2.3: LE910C1-EU c/ evaluation board

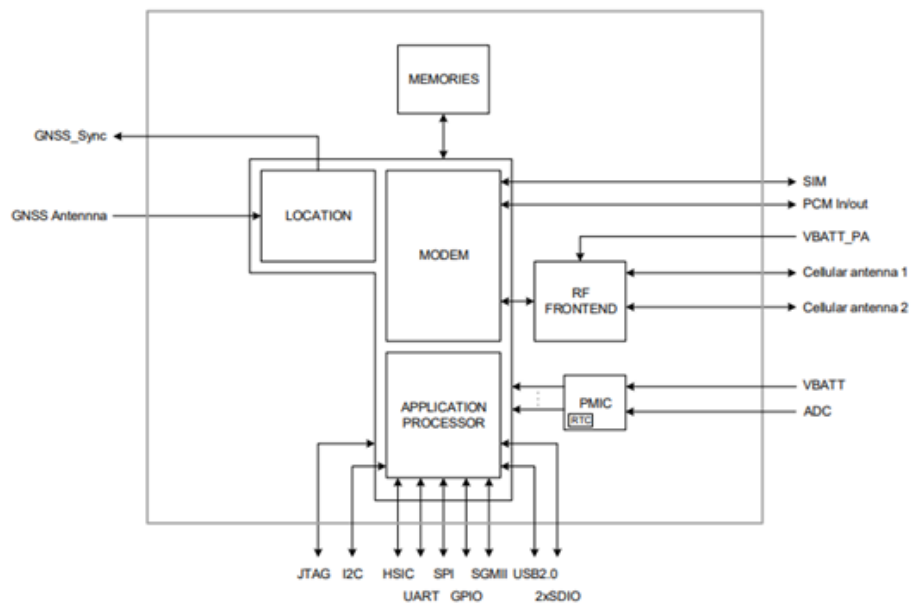


Figura 2.4: Diagrama de blocos do modem CAT1[3]

2.3 Modem NB-IOT - BC66

NB-IOT é uma tecnologia *low power wide area network (LPWAN)*, designada para dispositivos de baixa necessidade de largura de banda e funcionamento. Devido ao facto de ser uma tecnologia lançada recentemente, junho de 2016 (release 13 do 3GPP), não existe um suporte tão vasto e generalizado como acontece para o CAT1.

Apresenta velocidades de download na ordem dos 26 kbps, 66 kbps de upload e latências entre os 1,6 e 10s[11]. Ao contrário do CAT1, NB-IOT opera numa largura de banda bastante reduzida (180 KHz), uma proporção do espectro inutilizada.

A tecnologia NB-IOT é mais adequada para dispositivos estacionários (não suporta que a rede troque de torres), que apenas necessitem de enviar dados entre grandes intervalos de tempo, em que não interesse a latência. Oferece longo alcance e uma forte penetração de sinal, logo é bom para longas distâncias e uso interno ou subterrâneo[10].

Por exemplo em sensores de água, humidade, eletricidade, aplicativos de *smart city*, sensores de estacionamento, monitores industriais e sensores agrícolas.

O modem a testar será o BC66, Figura 2.5.

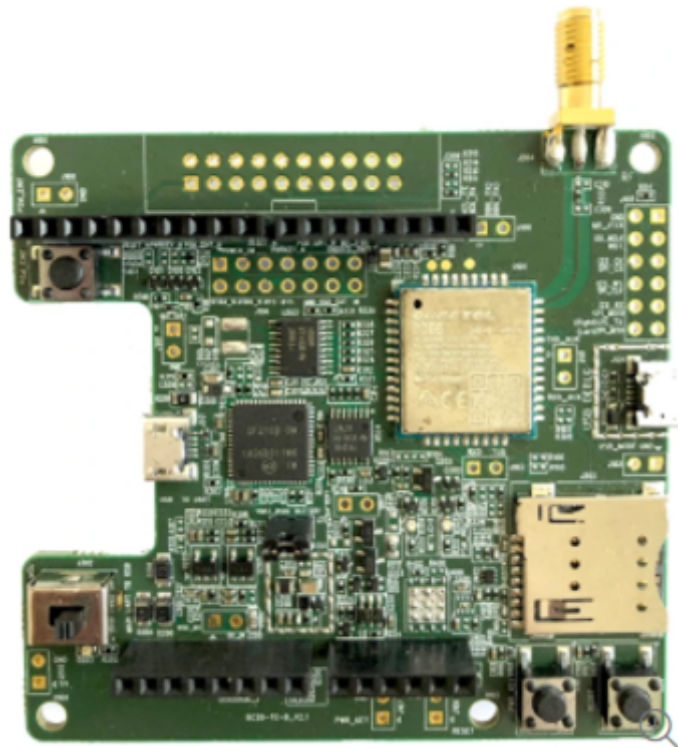


Figura 2.5: Quectel BC66[4]

2.4 Raspberry Pi

De modo controlar o modem na implementação final e correr código será utilizado um Raspberry Pi 3 Model B+, representado na Figura 2.6.



Figura 2.6: Raspberry Pi



Figura 2.7: Raspberry Pi c/ alimentação

Capítulo 3

Desenvolvimento do Projeto

3.1 Descodificação Mensagens

A fase inicial do projeto do dedicada a estudar e compreender o funcionamento do modem, tal como a comunicação com o computador, o controlo do modem, etc.

Após definir os parâmetros da Tabela 3.1, através do software fornecido pelo fabricante (telit AT Controller[12]), foi possível estabelecer conexão ao modem e enviar *Attention commands (AT Commands)*, e realizar operações básicas, tal como ligar para um telemóvel, atender chamadas, enviar SMS, Figura 3.1 e 3.2.

Tabela 3.1: Formato de Dados

Baud Rate	115200
Paridade	Nenhuma
Data bits	8
Stop bits	1
Flow Control	Manual

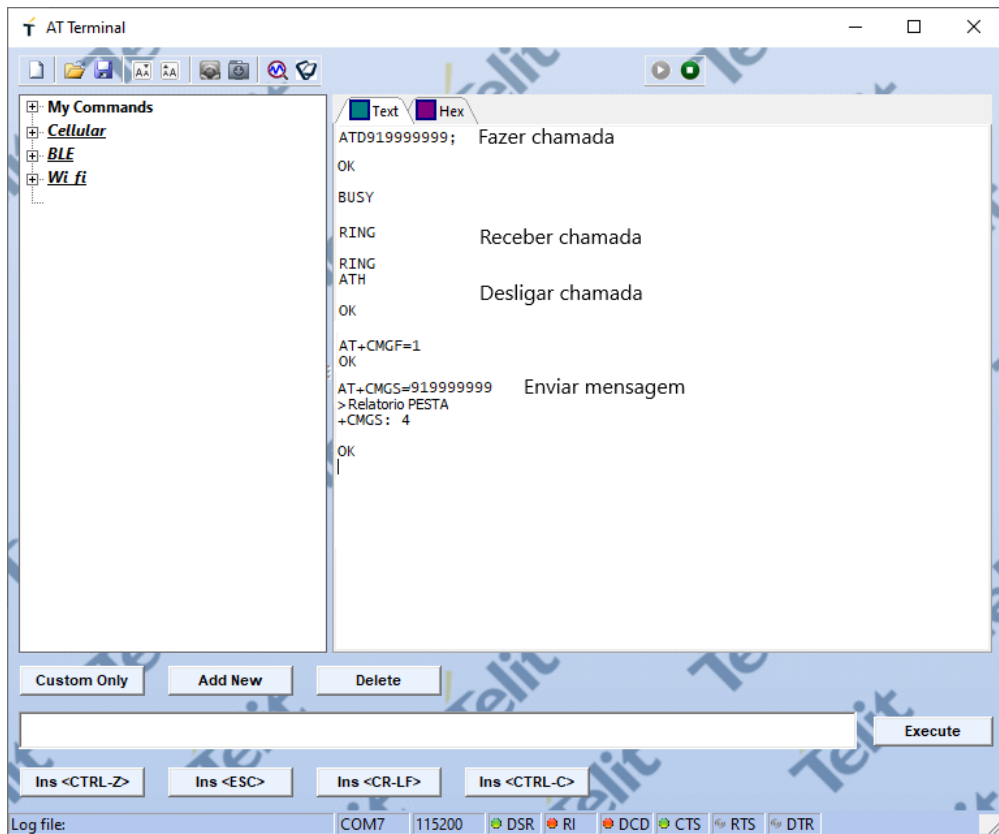
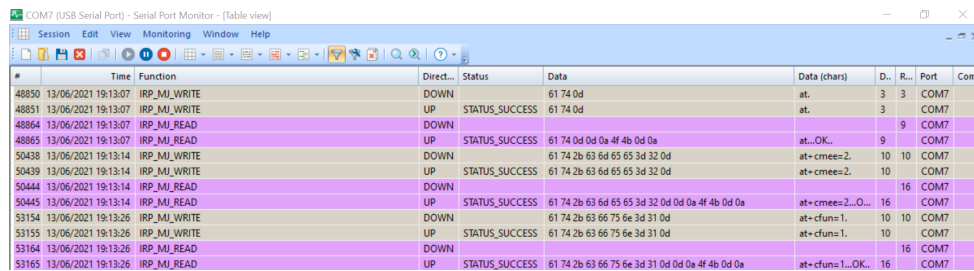


Figura 3.1: Teste comandos AT



Figura 3.2: Mensagem Recebida

Após enviar os dados para o modem, através de um *serial port sniffer*[13] foi possível ver como o software tratava os dados. Na Figura 3.3, após cada série de dados enviada, pode-se observar um elemento em comum, o caracter hexadecimal 0D, que traduz em *American Standard Code for Information Interchange (ASCII)* para *Carriage Return (CR)*, em linguagem C `"/r"`, esta informação será útil posteriormente, no desenvolvimento de *scripts* e da API.



#	Time	Function	Direct...	Status	Data	Data (chars)	D...	R...	Port	Comm
48850	13/06/2021 19:13:07	IRP_MJ_WRITE	DOWN		61 74 0d	at.	3	3	COM7	
48851	13/06/2021 19:13:07	IRP_MJ_WRITE	UP	STATUS_SUCCESS	61 74 0d	at.	3	3	COM7	
48864	13/06/2021 19:13:07	IRP_MJ_READ	DOWN				9	9	COM7	
48865	13/06/2021 19:13:07	IRP_MJ_READ	UP	STATUS_SUCCESS	61 74 0d 0d 0a 4f 4b 0d 0a	at...OK.	9	9	COM7	
50438	13/06/2021 19:13:14	IRP_MJ_WRITE	DOWN		61 74 2b 63 6d 65 65 3d 3d 0d	at+cmees=2.	10	10	COM7	
50439	13/06/2021 19:13:14	IRP_MJ_WRITE	UP	STATUS_SUCCESS	61 74 2b 63 6d 65 65 3d 3d 0d	at+cmees=2.	10	10	COM7	
50444	13/06/2021 19:13:14	IRP_MJ_READ	DOWN				16	16	COM7	
50445	13/06/2021 19:13:14	IRP_MJ_READ	UP	STATUS_SUCCESS	61 74 2b 63 6d 65 65 3d 3d 0d 0a 4f 4b 0d 0a	at+cmees=2...0...	16	16	COM7	
53154	13/06/2021 19:13:26	IRP_MJ_WRITE	DOWN		61 74 2b 63 66 75 6e 3d 31 0d	at+cfuns=1.	10	10	COM7	
53155	13/06/2021 19:13:26	IRP_MJ_WRITE	UP	STATUS_SUCCESS	61 74 2b 63 66 75 6e 3d 31 0d	at+cfuns=1.	10	10	COM7	
53164	13/06/2021 19:13:26	IRP_MJ_READ	DOWN				16	16	COM7	
53165	13/06/2021 19:13:26	IRP_MJ_READ	UP	STATUS_SUCCESS	61 74 2b 63 66 75 6e 3d 31 0d 0a 4f 4b 0d 0a	at+cfuns=1...OK...	16	16	COM7	

Figura 3.3: Serial Port Monitor

3.2 Ligação TCP

De forma a estabelecer ligação entre modem e servidor é preciso fazer uma ligação TCP, contudo antes de proceder na criação da mesma é preciso criar e configurar um servidor no computador pessoal para fazer a ligação ao modem é preciso configurar as definições do *Windows* e do router, por fim falta configurar as definições corretas do modem através de comandos AT.

3.2.1 Servidor

De maneira a criar um servidor no computador pessoal para fazer a ligação ao modem é preciso configurar as definições da firewall do *Windows* e do router.

Primeiro passo é criar uma regra na firewall do *Windows* para a porta neste caso 5000, na Figura 3.4 e 3.5 está uma breve visualização do procedimento.

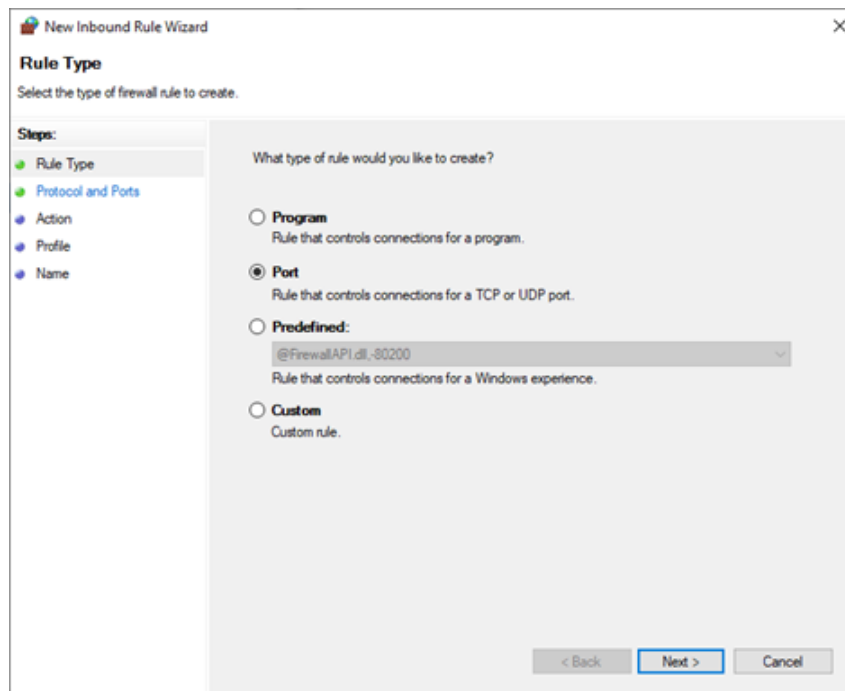


Figura 3.4: Regra Firewall(1)

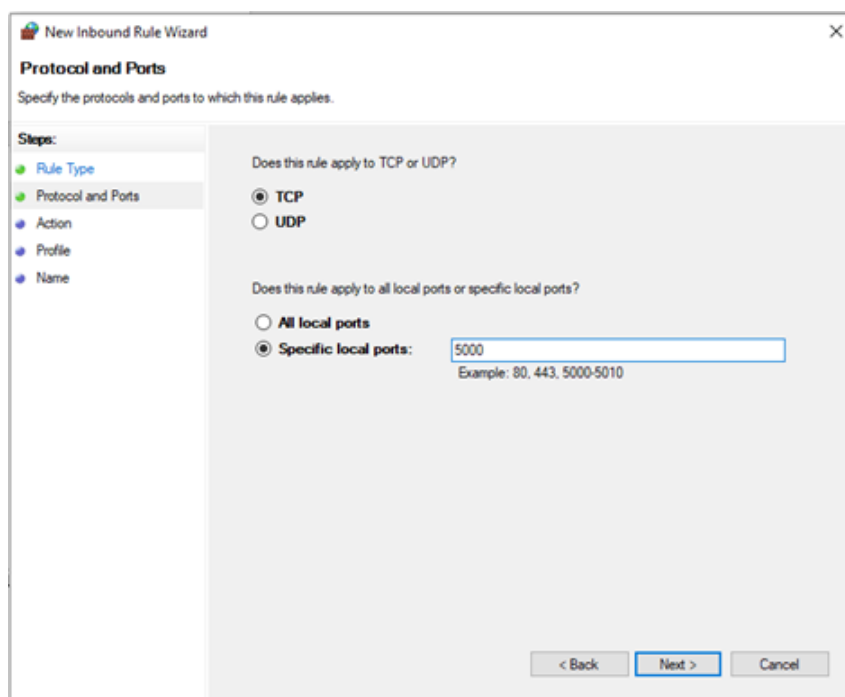


Figura 3.5: Regra Firewall(2)

Após as configurações no Windows, é necessário “abrir as portas” na página do router, Figura 3.6.

Novo

Apagar

	Nome de aplicação	Nome da WAN	Anfitrião interno	Anfitrião externo	Ativar
<input type="checkbox"/>	ServerTCP	Internet	192.168.1.67	--	Ativar

Tipo:

☒ Definido pelo utilizador
 ☐ Aplicação

Aplicação:

Selecione...

Ativar aplicação de portas:

☒

Nome de aplicação:

ServerTCP

Nome da WAN:

Internet

Anfitrião interno:

192.168.1.67

PC

Endereço IP externo de origem:

Protocolo:

TCP

Número da porta interna:

5000

5000

Número da porta externa:

5000

5000

Número da porta da fonte externa:

5000

5000

Apagar

Novo

Aplicar

Cancelar

Figura 3.6: Configuração Porta

Com as definições prontas, através da aplicação Hercules[14] a escutar a porta 5000 e a fazer *echo* aos dados recebidos, Figura 3.7, falta apenas configurar o modem e estabelecer a ligação.

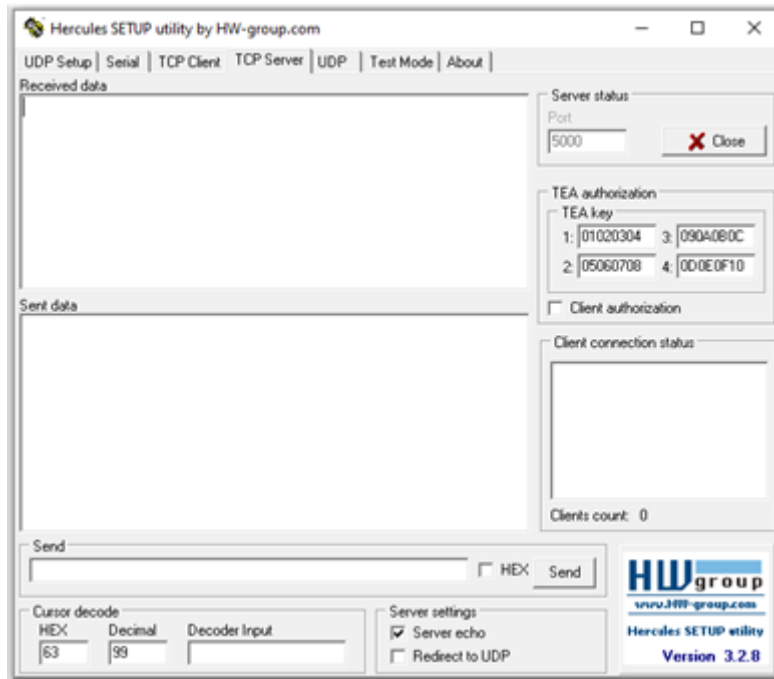


Figura 3.7: Servidor TCP - Hercules

3.2.2 One Time Setup

A série de comandos AT a seguir descrita é apenas necessária fazer uma vez no modem, estas configuram o contexto com a APN correta, as definições da socket e da ligação TCP.

- AT+CGDCONT=2,"IP","internet","",0,0,0,0 -> Selecionar o contexto 2 e selecionar o protocolo "IP" e definir a APN, "internet", este pode variar consoante a rede, se for Vodafone ="internet.vodafone.pt".
- AT#SCFGEXT=5,1,0,0,0,0 -> Seleciona a socket 5 e ativa o relatório do número de bytes de dados recebidos, quando recebe uma mensagem numa ligação TCP.
- AT#SCFGEXT2=5,0,0,0,0,2 -> Seleciona a socket 5 e ativa o relatório da causa de uma ligação TCP termine.
- AT#SCFG=5,2,1500,0,600,0 -> Seleciona a socket 5 e no contexto 2, define o pacote máximo de dados para o protocolo TCP/IP em 1500, desativa o timeout se não houver dados enviados, se não estabelecer uma ligação a um sistema remoto em 600 milissegundos dá timeout.

3.2.3 Initial Setup

A série de comandos AT a seguir descrita é necessária fazer sempre que se liga o modem, estas desbloqueiam o cartão SIM, ativa o *reporting* de erros e ativa o indicador de eventos quando o modem está em modo PSM.

- AT+CPIN=0000 -> Desbloquear o cartão SIM
- AT+CMEE=2 -> Ativa a notificação de erros em forma verbal.
- AT#PSMRI=500 -> Ativa a linha RI com duração de pulso 500 milissegundos.

3.2.4 TCP

No modem CAT1 é possível estabelecer dois tipos de ligação TCP, *command mode* e *online mode*, na primeira durante a ligação TCP é possível enviar *AT commands* e controlar o modem, enquanto que em *online mode*, após feita a ligação todos os dados enviados para o modem serão redirecionados para o servidor. Para voltar a poder controlar o modem seria necessário suspender a ligação, através do envio de "+++".

Na Figura 3.8 é possível ver todo o processo de uma ligação TCP em *command mode*, de ativar o contexto 2, abrir a ligação TCP ao servidor na socket 5. Após feita a conexão, pode-se enviar dados através do comando AT#SENDEXT=5,n.

Devido a definição de echo do servidor, o modem notifica imediatamente de que recebeu uma resposta através do "SRING:5, 3". Seguidamente é possível ler a resposta do servidor através do comando AT#SRECV=5,n.

Por fim para desligar a conexão basta fechar o socket e desativar o contexto.

- AT#SGACT=2,1 -> Ativa o contexto 2.
- AT#SD=5,0,XXXX,"YYY.YYY.YYY.YYY",0,0,1 -> Fazer ligação TCP na socket 5 ao servidor com porta = XXXX e IP= YYY.YYY.YYY.YYY, em modo comando.
- AT#SENDEXT=5,n -> Enviar n bytes para a ligação TCP na socket 5.
- SRING=5,n -> Mensagem de notificação do modem quando recebe n bytes de dados na ligação TCP na socket 5.
- AT#SRECV=5,n -> Ler a n bytes recebidos na socket 5.
- AT#SH=5 -> Fechar ligação na socket 5.
- AT#SGACT=2,0 -> Fechar o contexto 2.

```

at
OK
at+cpin=0000
OK
at+cme=2
OK
at#psmri=500
OK
at#sgact=2,1
#SGACT: 10.78.82.148
OK
at#sd=5,0,5000,"161.230.159.26",0,0,1
OK
at#ssendext=5,3
> abc
OK
SRING: 5,3
at#srecv=5,3
#SRECV: 5,3
abc
OK
at#sh=5
OK
at#sgact=2,0
OK

```

Figura 3.8: Ligação TCP

3.3 Power Save Mode

De modo a poupar o máximo de energia possível, é necessário meter o modem em modo PSM, seguindo o diagrama de blocos representado na Figura 3.9 e a Figura 3.10, é necessário enviar o código AT+CFUN=5, através do controlo manual desligar a linha *data terminal ready* (DTR), quando a linha *clear to send* (CTS) se desligar.

De modo a sair do PSM, basta fazer o inverso, ligar a linha DTR e enviar AT+CFUN=1.

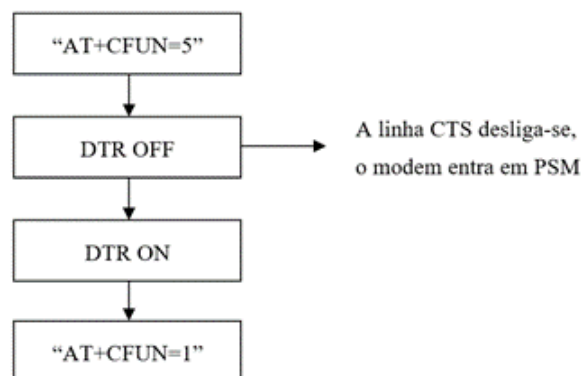


Figura 3.9: Diagrama de blocos PSM



Figura 3.10: Modo PSM

3.4 Recolher Latências e Consumos - CAT1

A fase seguinte do projeto foi dedicada, a avaliar os modems CAT1 e NB-IOT de forma a ter valores para comparar e poder seleccionar o modem mais acertado para o contexto em que será implementado.

De maneira a avaliar os modems será preciso criar primeiro *scripts* para facilitar a comunicação com os modems.

Após criados os *scripts*, serão apuradas as latências de comunicação e os consumos energéticos.

3.4.1 Scripts - Docklight

Para construir os scripts e potencializar o manuseio do modem será utilizado o software *Docklight Scripting*[15], estes scripts serão posteriormente a base para desenvolver a API.

Este software permite comunicar com o modem através da porta série, enviar comandos pré definidos, Figura 3.11 e 3.12 , controlo das linhas DTR e RTs e funcionar como servidor.

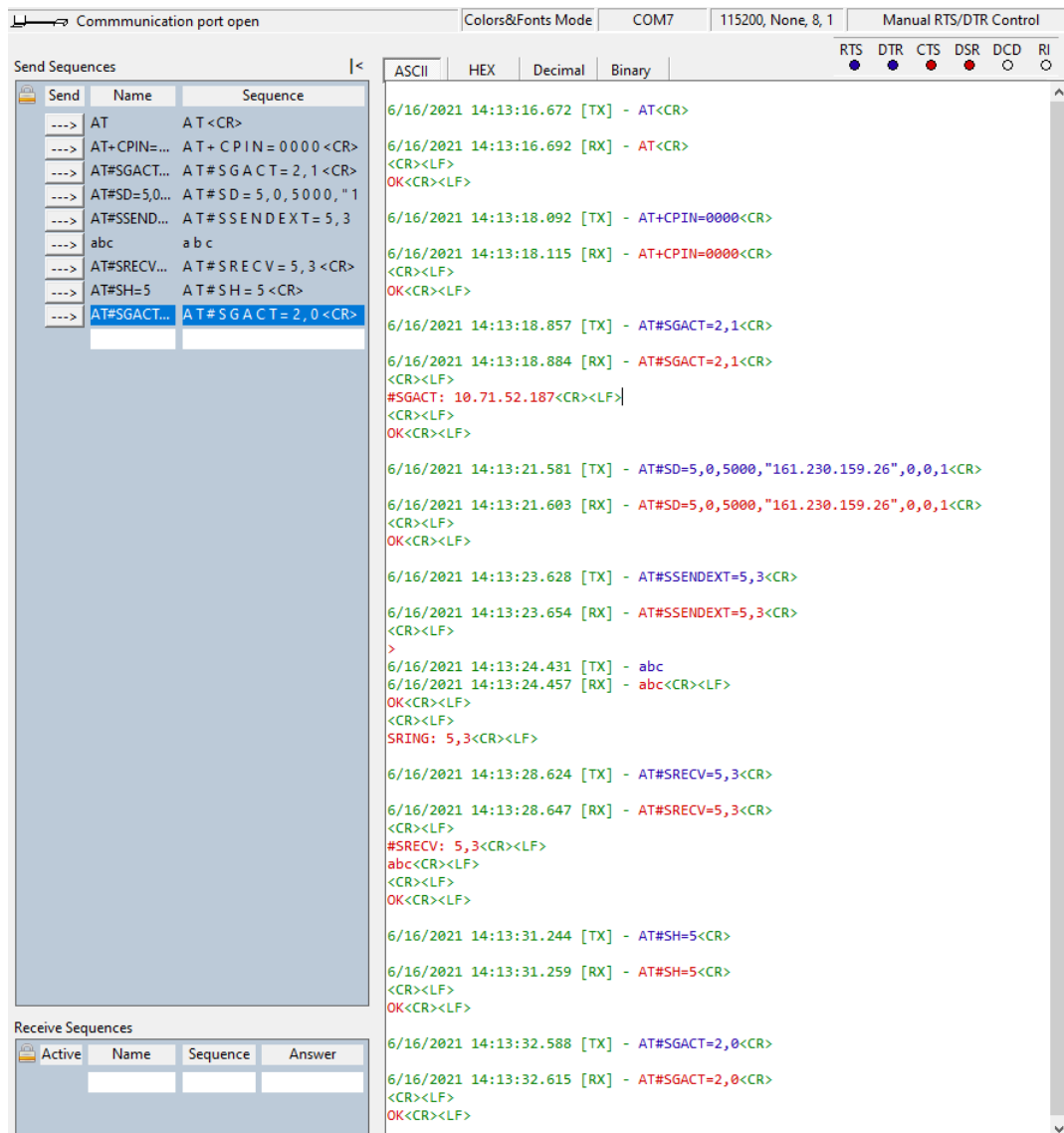


Figura 3.11: Modem Docklight Cliente

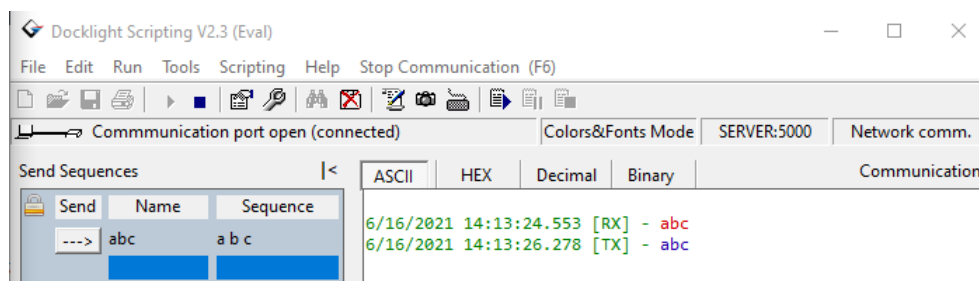


Figura 3.12: Docklight Servidor

3.4.2 Latências - CAT1

Com uma ligação TCP estabelecida, em *command mode* procedeu-se ao teste das latências, através do *Dockligh*, em grupos de 5 enviou-se 1, 10, 100, 1000 bytes.

De modo a manter os dados retirados os mais precisos possível, utilizou-se dois processos do Docklight no mesmo sistema, um com a ligação ao modem e outro com o servidor.

Recorrendo ao *Excel*, Tabela 3.2, foi calculada a latência usando os *timestamps* obtidos do Docklight.

Após retirados todos os dados denotou-se que o tamanho dos pacotes de dados enviados não surgia efeito no tempo em que o modem demorava a receber todos os dados, tendo todos valores semelhantes, resultando numa latência média total de 276 milissegundos.

Tabela 3.2: Latência CAT1

LATENCY				
COMMAND MODE		Server -» Client (Modem)		
Data Payload	Measure ID	Timestamp TX	Timestamp RX	Latency (sec)
1 Byte	1	09:51:18.301	09:51:18.513	00:00:00.212
	2	09:51:19.743	09:51:20.049	00:00:00.306
	3	09:51:20.833	09:51:20.992	00:00:00.159
	4	09:51:21.832	09:51:21.969	00:00:00.137
	5	09:51:22.917	09:51:23.232	00:00:00.315
	Average			00:00:00.226

3.4.3 Consumos - CAT1

Para medir apenas os consumos energéticos do módulo LE910C1-EU, foi necessário alimentar o modem com uma fonte externa de 3,9 V, seguindo o *datasheet*[16] foi preciso reposicionar os *jumpers* inicialmente na posição a verde para a posição vermelha, Figura 3.13 e 3.14 .

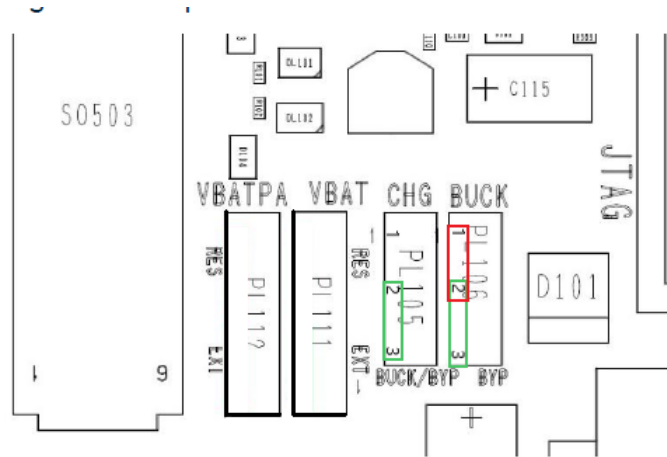


Figura 3.13: Jumpers PL105/PL106

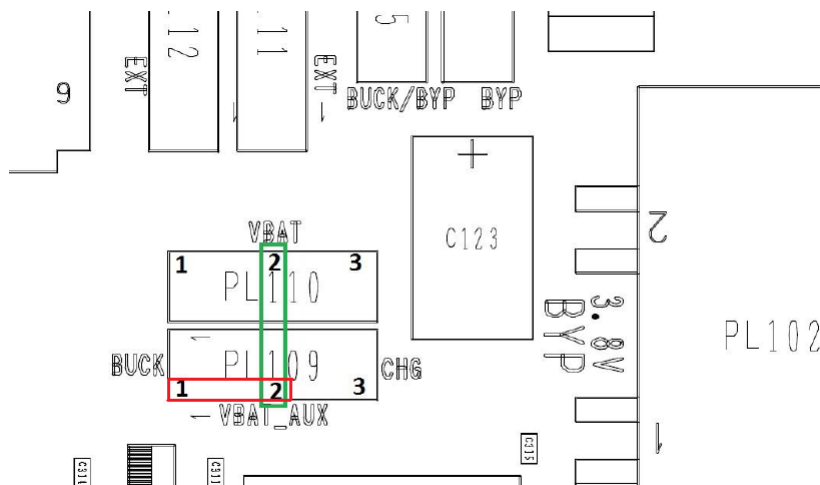


Figura 3.14: Jumpers PL109/PL110

Com os *jumpers* na posição correta ligou-se a fonte externa a um amperímetro e um *switch* fechado em paralelo, Figura 3.15, a necessidade deste *switch* advém do facto do modem não conseguir ligar com o amperímetro em série. Após o modem ligado e funcional, abre-se o *switch* e mede-se a corrente.

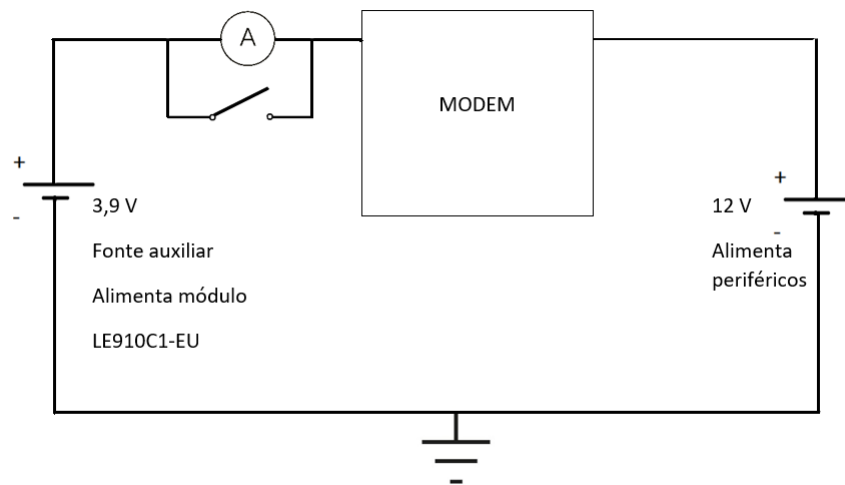


Figura 3.15: Esquema Alimentação

Foram medidos os consumos em *power save*, *idle* e em transferência de dados e posteriormente anotados em Excel, Tabela 3.3.

O modem em modo normal, "AT+CFUN=1", em *idle* apresentava valores de corrente na ordem dos 9,5 mA, com picos de 150 mA. Durante as comunicações com o modem, enviar comandos *AT*, receber/enviar dados na ligação TCP, a corrente chegava a picos de 150 mA. Em modo *PSM*, "AT+CFUN=5", o consumo de corrente desceu para os 2,33 mA, sem picos e em comunicações mantêm-se os 150 mA anteriormente registados.

Devido a negociações com a rede, cada vez que o modem tenta entrar em *PSM*, demora à volta de 9 segundos, isto deve-se a negociações com a rede, este tempo pode ser reduzido pelo fornecedor do cartão SIM.

Tabela 3.3: Consumo CAT1

Power Consumption			
Mode		Voltage (V)	Current (mA)
Full functionality (CFUN=1)	IDLE	3.9	9.5
	Data Transfer	3.9	150
Power saving (CFUN=5)	IDLE	3.9	2.33
	Data Transfer	3.9	150

3.5 NBIOT

No modem de NB-IOT, foram realizados os mesmos testes, latências e consumos, foi necessária a adaptação dos *scripts*, pois para além dos comandos AT mais básicos, os restantes podem não ser iguais de fornecedor para fornecedor.

3.5.1 Latências - NB-IOT

O procedimento estabelecido para o modem CAT1 foi repetido e enviado 5 grupos de 1, 10, 100 e 1000 bytes, retirados os *timestamps*, anotados e tratados no *Excel*, Tabela 3.4.

A latência média total calculada foi 2,72 segundos, estando dentro do intervalo estudado anteriormente no capítulo 2.3.

Tabela 3.4: Latência NB-IOT

LATENCY				
COMMAND MODE		Server -» Client (Modem)		
Data Payload	Measure ID	Timestamp TX	Timestamp RX	Latency (sec)
1 Byte	1	11:10:44.119	11:10:47.406	00:00:03.287
	2	11:11:49.726	11:11:52.793	00:00:03.067
	3	11:12:58.101	11:12:59.231	00:00:01.130
	4	11:13:29.907	11:13:34.211	00:00:04.304
	5	11:14:05.651	11:14:09.254	00:00:03.603
	Average			00:00:03.078

3.5.2 Consumos - NB-IOT

Seguindo o mesmo método que foi utilizado no modem CAT1, tentou-se medir os consumos do módulo BC66, contudo devido à resistência interna do amperímetro não foi possível ler nenhum valor de corrente, pois assim que se abria o *switch* o modem desligava. Foi testado também o uso de um osciloscópio mas apresentava muito ruído o que tornava impossível a leitura.

Após várias tentativas sem sucesso de medir a corrente, deu-se por inconclusivo, seguindo-se então pelos valores apresentados no *datasheet*[17] de 3.5 μ A para o estado de PSM.

PSM e eDRX

Ao contrário do modem CAT1, é possível entrar no modo PSM relativamente rápido, através da configuração do comando "AT+CPSMS"[18].

Neste comando é possível definir o valor de 2 *Timers*, T3412 e T3324, Figura 3.16.

O T3412, determina o intervalo de tempo em que o modem, está em *sleep* antes de se conectar à rede para fazer a verificação de dados.

O T3324, define o tempo para o modem voltar a entrar no modo PSM, após deixar de comunicar e entrar em *idle*.

O valor destes *Timers*, pode ser calculado recorrendo ao *datasheet*, os bits 8 a 6 são o multiplicador e os bits 5 a 1 são o valor que será multiplicado, ou através de *websites* dedicados ao cálculo destes *Timers*[19].

Exemplo Cálculo T3324

Para definir o tempo que o modem demora a entrar em *sleep* para 4 minutos: 3.1

- Temos de definir o multiplicador para 1 minuto: 001
- Definir o últimos bits para o valor 4: 00100

$$T3324 = 00100100. \quad (3.1)$$

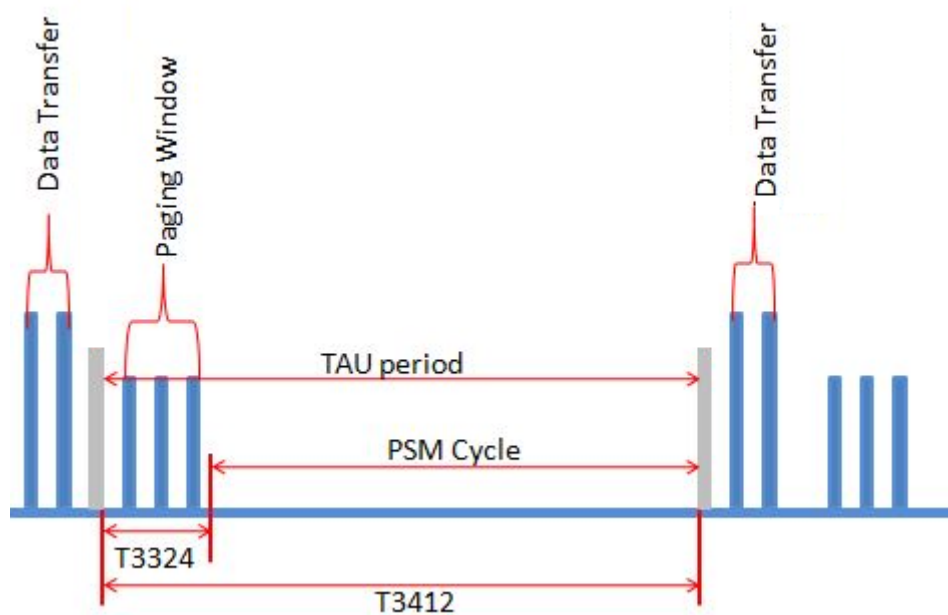


Figura 3.16: T3412 e T3324[5]

Com o *release* 13 do 3GPP, foi introduzido o conceito de *Extended Discontinuous Reception*(eDRX) de modo a auxiliar os mecanismos já existentes de PSM, sendo que estes podem ser utilizados em simultâneo. Ao invés do PSM com o eDRX, o modem pode escutar notificações de dados pendentes sem ter de estabelecer uma ligação completa à rede[20].

O eDRX é um bom protocolo para dispositivos IOT, pois consegue acordar mais rápido que o PSM, verificar se existe dados pendentes e suportar *paging cycles* maiores, permitindo assim consumir menos energia[20]. Na Figura 3.17, está representado a comparação entre PSM e eDRX.

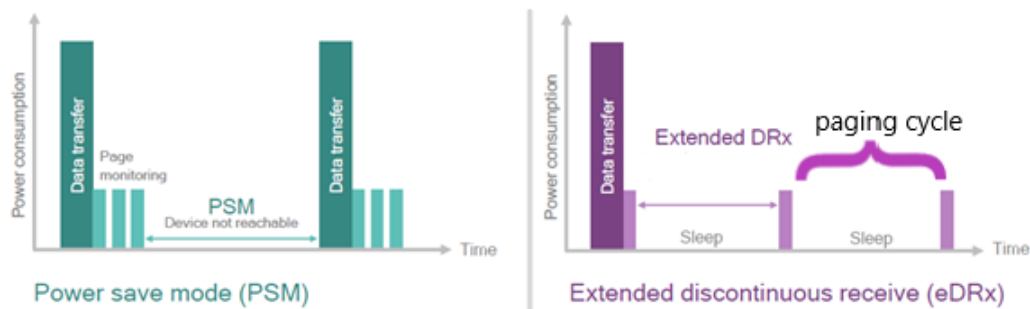


Figura 3.17: PSM e eDRX[6]

3.6 Resultados Obtidos

Com as latências calculadas e os consumos energéticos retirados, comparou-se os resultados obtidos de ambos os modems para escolher aquele que se iria adequar melhor às necessidades da implementação projetada pela DMS.

3.6.1 Latências CAT1 vs NB-IOT

O modem CAT1, apresentou latências bastante competitivas, com uma média de 0,276 segundos.

Por outro lado o modem NB-IOT com latências médias significativamente maiores, 2,72 segundos, não sendo o ideal.

3.6.2 Consumos CAT1 vs NB-IOT

O modem CAT1 registou consumos mais baixos que o esperado, com 2,33 mA em modo PSM, contudo possui o *delay* de 9 segundos antes de conseguir entrar em PSM.

Como não foi possível determinar o consumo do modem NB-IOT, apenas se ficou com a referência do *datasheet*, 3.5 μ A, não podendo assim fazer uma avaliação do consumo real.

Face aos resultados obtidos e às necessidades do projeto, optou-se por seguir o projeto com a tecnologia CAT1, devido às baixas latências e consumos porém maiores que o modem NB-IOT, no entanto baixos.

Capítulo 4

Implementação da API

Na Figura A.1 está representado o fluxograma geral da API desenvolvida, ao longo deste capítulo serão apresentadas as funções concebidas e uma breve descrição de cada uma, para uma análise mais detalhada, a listagem do código devidamente comentado, pode ser consultada em anexo A.

A plataforma de desenvolvimento utilizada foi o *Visual Studio Code*, com a extensão SFTP, para atualizar o código no Raspberry Pi à medida que ia sendo escrito.

O software encontra-se dividido em duas partes, a parte da função menu e da *main*, implementadas para testar o desenvolvimento da API e direcionadas para o *Debug* e a segunda parte, um conjunto de funções que permite abrir a porta série ligada ao modem, enviar comandos AT, estabelecer ligação TCP a um servidor e entrar em PSM.

4.1 Função ligação ao modem

No Linux, tudo são ficheiros, logo para estabelecer ligação à porta série, basta fazer uso da função *"open"* e abrir o ficheiro *"/dev/ttyUSB0"*, definir a paridade, *stop bits*, *data bits* e *baud rate*. Após abrir o ficheiro há a verificação de erro ou confirmação de sucesso 4.1.

```
1 int OpenSerial()
2 {
3     int serial_port = open("/dev/ttyUSB0",
4                             O_RDWR);
5
6     if (serial_port == -1) /* Error Checking
7                             */
8     {
9         #if DEBUG == 1
10            printf("\n Error! in Opening ttyUSB0
11                  ");
12        #endif
13        return -1;
14    }
15    else
16    {
17        #if DEBUG == 1
18            printf("\n ttyUSB0 Opened
19                  Successfully ");
20        #endif
21    }
22    .
23    .
24    tty.c_cflag &= ~PARENB; // Clear parity
25                           bit, disabling parity
26    tty.c_cflag &= ~CSTOPB; // Clear stop
27                           field, only one stop bit used in
28                           communication
29    tty.c_cflag &= ~CSIZE; // Clear all bits
30                           that set the data size
31    tty.c_cflag |= CS8;     // 8 bits per
32                           byte
33    .
34    cfsetispeed(&tty, B115200);
35    cfsetospeed(&tty, B115200);
36    .
37    return serial_port;
38 }
```

Listagem 4.1: Abrir Porta Série

4.2 Função enviar dados

A fim de enviar dados para o modem, basta escrever no ficheiro aberto. A função que permite a escrita de dados para a porta série, baseia-se no uso da função *write*, tendo como parâmetros a variável "serial_port", associada à porta série, o buffer "msg" de dados a enviar e o correspondente tamanho em bytes. Posteriormente é feita a verificação de erros 4.2.

```
1 write(serial_port, msg, strlen(msg));
```

Listagem 4.2: Escrever Dados

4.3 Função ler dados

A função que permite a leitura de dados da porta série, baseia-se no uso da função *read*, tendo como parâmetros a variável "serial_port", associada à porta série, o buffer "read_buf" de dados a receber e o correspondente tamanho em bytes. Posteriormente é feita a verificação de erros e a impressão da resposta do modem 4.3.

```
1 read(serial_port, &read_buf, sizeof(read_buf)
    );
2 printf("%s\n", read_buf);
```

Listagem 4.3: Ler Dados

4.4 Função escrever comandos

A função *SendCommand()* permite ao utilizador escrever os comandos AT para o buffer "msg" que deseja enviar no terminal. A função adiciona automaticamente o *carriage return* no fim de cada comando que o utilizador escreva. A função termina quando o utilizador mandar "back" e volta ao menu principal 4.4

```
1 void SendCommand()  
2 {  
3     while (strcmp(msg, "back\r") != 0 &&  
4             strcmp(msg, "BACK\r") != 0)  
5     {  
6         printf("\nCODE:");  
7         scanf("%s", msg);  
8         strncat(msg, "\r", 2);  
9         ReadWrite(msg);  
10    }
```

Listagem 4.4: Escrever Comandos AT

4.5 Função Ler/Escriver

A função principal de comunicação com o modem é responsável por receber o *buffer* com o comando AT, fazer a verificação se o modem está em PSM, ativar a linha DTR, enviar o comando para o modem e voltar a desativar a linha 4.5.

```
1  
2 void ReadWrite(char msg[])  
3 {  
4     int i, o = 0;  
5  
6     if (FLAG_DTR == 0)  
7     {  
8         SetDTR();  
9     }  
10  
11    CheckCFUN(msg);  
12    i = SendData(msg);  
13    o = ReadData(msg);  
14  
15    if (FLAG_DTR == 0)  
16    {  
17        ClearDTR();  
18    }  
19 }
```

Listagem 4.5: Ler e Escrever

4.6 Verificar PSM

A função é responsável cada vez que se envia uma mensagem por verificar se foi alterado o modo de funcionamento do modem, PSM ou normal 4.6.

Caso o comando seja "AT+CFUN=1", a "FLAG_DTR" é definida a 1 e a linha DTR ligada 4.7.

Caso o comando seja "AT+CFUN=5", a "FLAG_DTR" é definida a 0 e a linha DTR desligada 4.8

```
1
2 void CheckCFUN(char msg[])
3 {
4     if (strcmp(msg, "at+cfun=5\r") == 0 ||
5         strcmp(msg, "AT+CFUN=5\r") == 0)
6     {
7         FLAG_DTR = 0;
8         ClearDTR();
9     }
10    if (strcmp(msg, "at+cfun=1\r") == 0 ||
11        strcmp(msg, "AT+CFUN=1\r") == 0)
12    {
13        FLAG_DTR = 1;
14        SetDTR();
15    }
```

Listagem 4.6: Check CFUN

4.6.1 Set DTR

```
1
2 void SetDTR()
3 {
4     int status;
5     ioctl(serial_port, TIOCMGET, &status); //
        GET DTR PIN status
6
7     status |= TIOCM_DTR;
8
9     ioctl(serial_port, TIOCMSET, status); //
        SET DTR PIN to 1
10
11 }
```

Listagem 4.7: Set DTR

4.6.2 Clear DTR

```
1 void ClearDTR()
2 {
3     int status;
4     ioctl(serial_port, TIOCMGET, &status); //
        GET DTR PIN status
5
6     status &= ~TIOCM_DTR;
7
8     ioctl(serial_port, TIOCMSET, status); //
        SET DTR PIN to 0
9
10 }
```

Listagem 4.8: Clear DTR

4.7 One Time Setup

Conjunto de comandos criados anteriormente 3.2.2 para enviar 4.9.

```
1 void OneTime()
2 {
3     char msg[10][50];
4
5     strcpy(msg[0], CGDCONT Context ", \"IP\"
6             ,\" \" APN \" \", \"\",0,0,0,0\r");
7     strcpy(msg[1], SCFGEXT Socket_TCP
8             SCFGEXT_N);
9     strcpy(msg[2], SCFGEXT2 Socket_TCP
10            SCFGEXT2_N);
11    strcpy(msg[3], SCFG Socket_TCP SCFG_N);
12
13    for (int i = 0; i < 4; i++)
14    {
15        ReadWrite(msg[i]);
16    }
17 }
```

Listagem 4.9: One Time Setup

4.8 Initial Setup

Conjunto de comandos criados anteriormente 3.2.3 para enviar no *startup* do modem 4.10.

```
1 void InitialSetup()
2 {
3     char msg[5][50];
4     strcpy(msg[0], AT);
5     strcpy(msg[1], CFUN "1\r");
6     strcpy(msg[2], CMEE);
7     strcpy(msg[3], PSMRI);
8     strcpy(msg[4], CPIN PIN "\r");
9
10    for (int i = 0; i < 5; i++)
11        ReadWrite(msg[i]);
12 }
```

Listagem 4.10: Initial Setup

4.9 Ligação TCP

Esta função ativa o contexto 2, e liga-se ao servidor com "IPadd" e "PORT" guardados em macros no ficheiro "main.h", A.2. Após feita a ligação, é feita a função "SendCommand()" para permitir o utilizador comunicar com o servidor 4.11.

```
1 void OpenSocketServer()
2 {
3     char msg[2][50];
4
5     strcpy(msg[0], SGACT SGACT_CONTEXT ",1\r"
6             );
7     strcpy(msg[1], SD Socket_TCP ",0," PORT "
8             ,\" IPadd "\",0,0,1\r");
9
10    ReadWrite(msg[0]);
11    ReadWrite(msg[1]);
12
13    SendCommand();
14 }
```

Listagem 4.11: Abrir ligação TCP

4.10 Fechar ligação TCP

Desliga a ligação TCP e desativa o contexto 2, 4.12.

```
1 void CloseSocketServer()
2 {
3     char msg[2][50];
4
5     strcpy(msg[0], SH Socket_TCP "\r");
6     strcpy(msg[1], SGACT SGACT_CONTEXT ",0\r"
7             );
8
9     ReadWrite(msg[0]);
10    ReadWrite(msg[1]);
11
12 }
```

Listagem 4.12: Fechar ligação TCP

4.11 Entrar em PSM

Envia o comando "AT+CFUN=5", 4.13

```
1 void EnterPSM()
2 {
3     char msg[50];
4
5     strcpy(msg, CFUN5);
6     ReadWrite(msg);
7 }
```

Listagem 4.13: Entrar em PSM

4.12 Sair PSM

Envia o comando "AT+CFUN=1", 4.14

```
1 void ExitPSM()
2 {
3     char msg[50];
4
5     strcpy(msg, CFUN1);
6     ReadWrite(msg);
7 }
```

Listagem 4.14: Sair PSM

4.13 Menu de opções

Menu de opções, de cada função a ser executada. Dá *return* à escolha do utilizador, para na main correr a função, 4.15.

```
1 char menu()
2 {
3     char choice = ' ';
4     do
5     {
6         system("clear");
7         printf("flag dtr = %d\n", FLAG_DTR);
8         printf("-----\n\n");
9         printf("1- Initial Setup\n");
10        printf("2- Start TCP Connection\n");
11        printf("3- Send Commands\n");
12        printf("4- Enter PSM Mode\n");
13        printf("5- Close TCP Connection\n");
14        printf("6- Exit PSM Mode\n");
15        printf("8- Restart Modem\n");
16        printf("9- One Time Setup\n");
17        printf("s/S- Exit.\n");
18        printf("-----\n\n");
19        scanf("%c", &choice);
20    } while (choice != 's' && choice != 'S'
21            && choice != '1' && choice != '2' &&
22            choice != '3' && choice != '4' &&
23            choice != '5' && choice != '6' &&
24            choice != '7' && choice != '8' &&
25            choice != '9');
26
27    return choice;
28 }
```

Listagem 4.15: Função Menu

4.14 Main

Na função main, é feita a conexão ao modem assim que o programa é iniciado, a seguir é feito um "fork", onde o filho corre constantemente a função de ler dados.

A função menu e um *switch case* para seleccionar a ação a realizar pelo modem.

Quando o *switch chase* recebe um "S" para sair e terminar o programa, encerra o socket da ligação TCP, desliga o modo PSM, fecha a ligação à porta série e termina o processo filho anteriormente criado, 4.16.

```
1  int main()
2  {
3      system("clear");
4
5      char choice = ' ';
6
7      serial_port = OpenSerial();
8
9      pid_t pid = fork();
10
11     if (pid == 0)
12     {
13         while (1)
14         {
15             ReadData();
16         }
17     }
18
19     do
20     {
21         choice = menu();
22
23         switch (choice)
24         {
25             case '1':
26             {
27                 system("clear");
28                 InitialSetup();
29                 break;
30             }
31
32             case '2':
33             {
34                 system("clear");
35                 OpenSocketServer();
36                 break;
37             }
38
```

```

39         case '3':
40         {
41             system("clear");
42             SendCommand(); //Write commands
43             break;
44         .
45         .
46         .
47         .
48         case '9':
49         {
50             OneTime();
51             SystemPause();
52             break;
53         }
54     }
55 } while (choice != 's' && choice != 'S');
56
57 CloseSocketServer();
58 ExitPSM();
59 close(serial_port);
60 kill(pid, SIGTERM);
61 return 0;
62 }
```

Listagem 4.16: Main

4.15 Macros

No ficheiro main.h estão presentes as bibliotecas necessárias, os protótipos de cada função utilizada no código e as macros definidas 4.17.

```

1
2
3
4 int serial_port;           //Serial Port
5 int FLAG_DTR;              //FLAG==1, dtr is on and modem is free to
    communicate
6                             //FLAG==0, Modem is in PSM
7
8
9 #define APN "internet"
10 #define IPadd "161.230.159.26"
11 #define PORT "5000"
12 #define PIN "0000"
13
14 //Restart
```

```

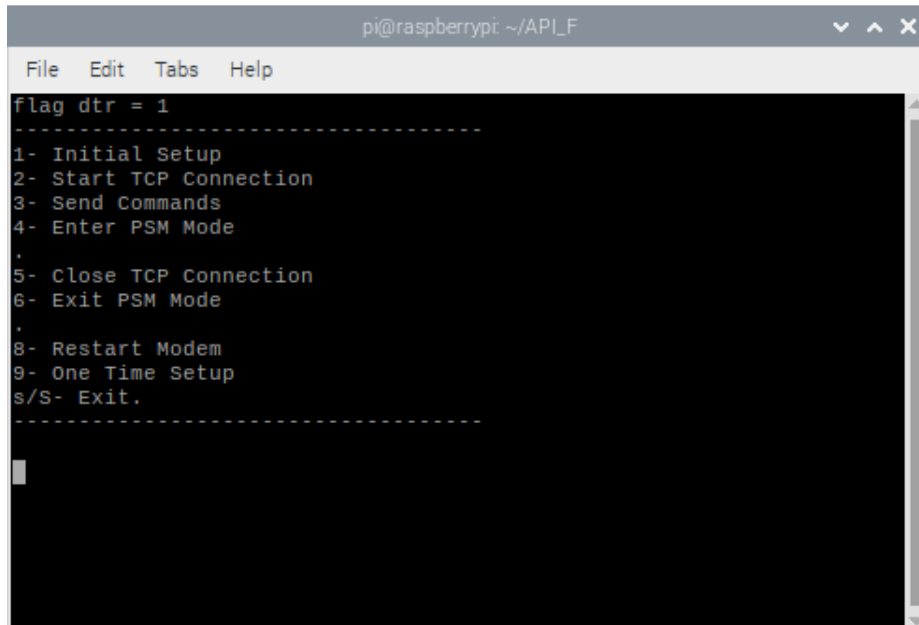
15 #define Reboot "AT#ENHRST=1,0\r"
16
17 //Initial Setup
18 #define AT "AT\r"
19 #define CMEE "AT+CMEE=2\r"          /// error report
20 #define CPIN "AT+CPIN="             /// SIM card pin
21 #define PSMRI "AT#PSMRI=500\r"      ///
22 #define CFUN "AT+CFUN="             /// PSM mode
23 #define CFUN5 "AT+CFUN=5\r"         /// Turn on PSM
24 #define CFUN1 "AT+CFUN=1\r"         /// Turn off PSM
25
26 //SocketServer
27 #define SGACT "AT#SGACT="            /// Context management
28 #define SGACT_CONTEXT "2"
29 #define SD "AT#SD="                 /// Start TCP Connection
30 #define SH "AT#SH="                 /// Close TCP connection
31 #define Socket_TCP "5"              /// Modem socket to open TCP
    connection
32
33 //OneTime                            /// Socket management
34 #define CGDCONT "AT+CGDCONT="
35 #define Context "2"
36 #define SCFGEXT "AT#SCFGEXT="
37 #define SCFGEXT_N " ,1,0,0,0,0\r"
38 #define SCFGEXT2 "AT#SCFGEXT2="
39 #define SCFGEXT2_N " ,0,0,0,0,2\r"
40 #define SCFG "AT#SCFG="
41 #define SCFG_N " ,2,1500,0,600,0\r"

```

Listagem 4.17: ficheiro Header

4.16 Exemplos de Resultados Obtidos

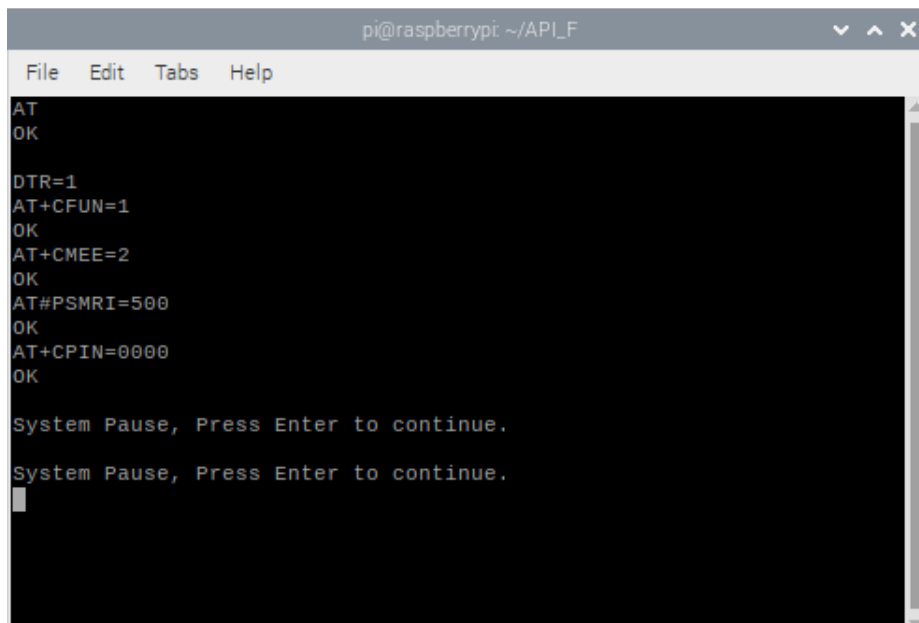
No início da aplicação é apresentado o menu com as várias opções propostas ao utilizador 4.1.

A screenshot of a terminal window titled 'pi@raspberrypi: ~/API_F'. The window contains a menu with the following options: '1- Initial Setup', '2- Start TCP Connection', '3- Send Commands', '4- Enter PSM Mode', '5- Close TCP Connection', '6- Exit PSM Mode', '8- Restart Modem', '9- One Time Setup', and 's/s- Exit.'. The menu is preceded by 'flag dtr = 1' and followed by a dashed line. A cursor is visible at the bottom left of the terminal.

```
pi@raspberrypi: ~/API_F
File Edit Tabs Help
flag dtr = 1
-----
1- Initial Setup
2- Start TCP Connection
3- Send Commands
4- Enter PSM Mode
.
5- Close TCP Connection
6- Exit PSM Mode
.
8- Restart Modem
9- One Time Setup
s/s- Exit.
-----
█
```

Figura 4.1: Menu API

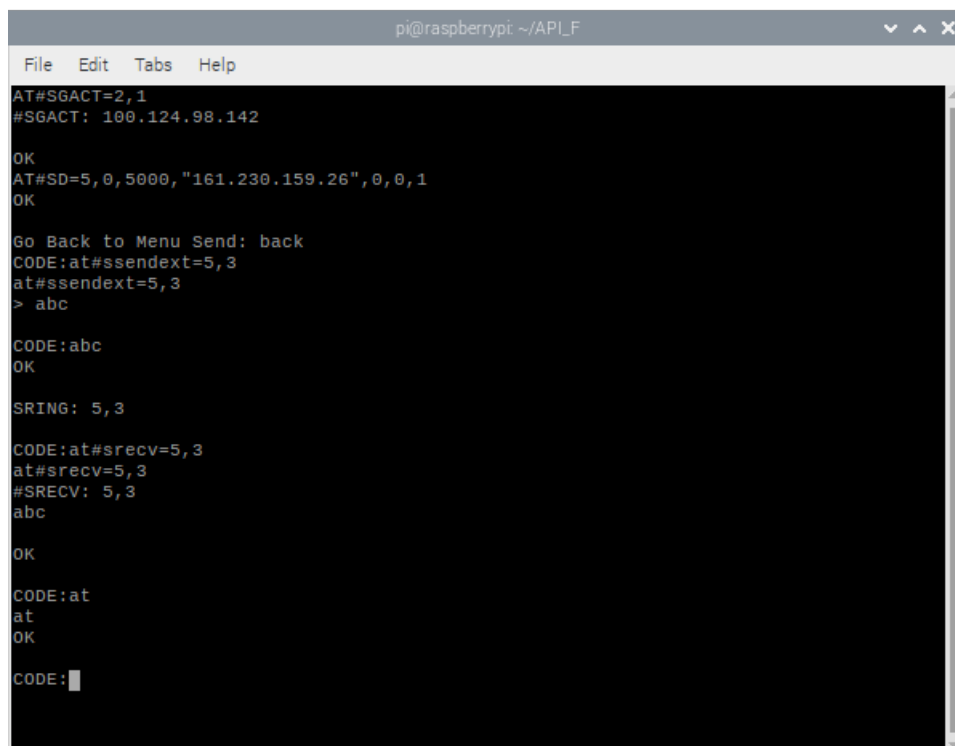
- Seguindo a numeração e fazendo o Initial Setup 4.2.

A screenshot of a terminal window titled 'pi@raspberrypi: ~/API_F'. The window shows the output of the 'Initial Setup' command, including AT commands and their responses: 'AT', 'OK', 'DTR=1', 'AT+CFUN=1', 'OK', 'AT+CMEE=2', 'OK', 'AT#PSMRI=500', 'OK', 'AT+CPIN=0000', 'OK'. It also shows two 'System Pause, Press Enter to continue.' messages. A cursor is visible at the bottom left of the terminal.

```
pi@raspberrypi: ~/API_F
File Edit Tabs Help
AT
OK
DTR=1
AT+CFUN=1
OK
AT+CMEE=2
OK
AT#PSMRI=500
OK
AT+CPIN=0000
OK
System Pause, Press Enter to continue.
System Pause, Press Enter to continue.
█
```

Figura 4.2: Initial Setup API

- Iniciar ligação TCP 4.3.



```
pi@raspberrypi: ~/API_F
File Edit Tabs Help
AT#SGACT=2,1
#SGACT: 100.124.98.142

OK
AT#SD=5,0,5000,"161.230.159.26",0,0,1
OK

Go Back to Menu Send: back
CODE:at#ssendext=5,3
at#ssendext=5,3
> abc

CODE:abc
OK

SRING: 5,3

CODE:at#srecv=5,3
at#srecv=5,3
#SRECV: 5,3
abc

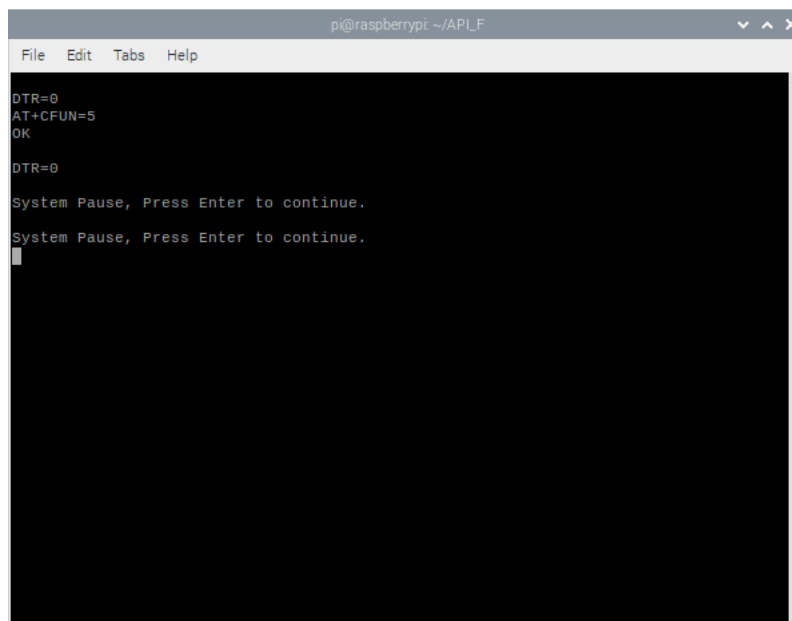
OK

CODE:at
at
OK

CODE:█
```

Figura 4.3: Ligação TCP API

- Enter PSM 4.4.



```
pi@raspberrypi: ~/API_F
File Edit Tabs Help

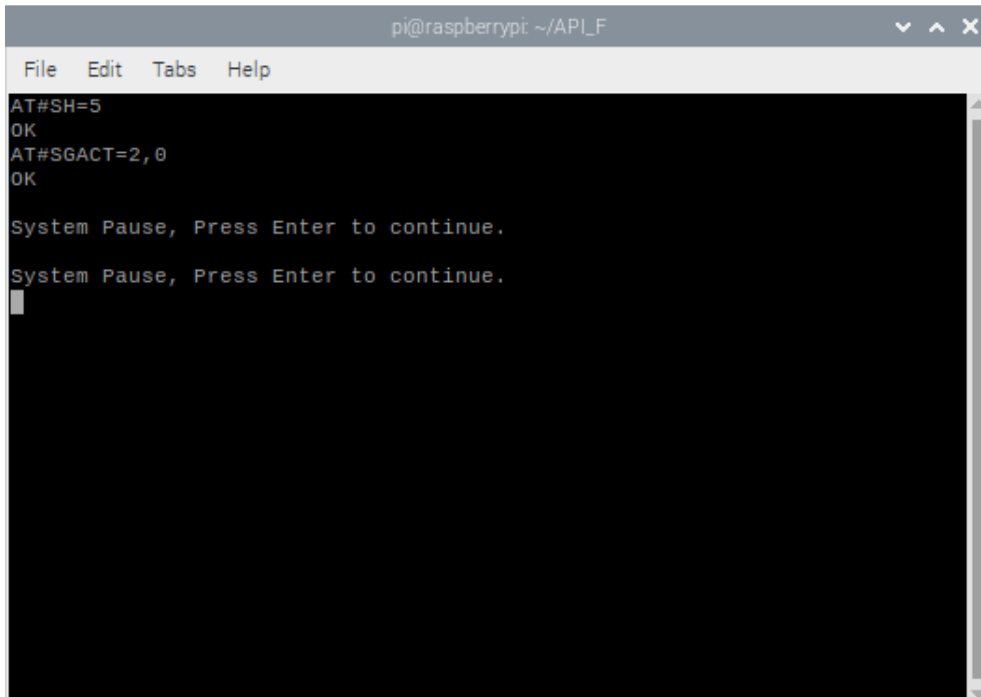
DTR=0
AT+CFUN=5
OK

DTR=0

System Pause, Press Enter to continue.
System Pause, Press Enter to continue.
█
```

Figura 4.4: Enter PSM API

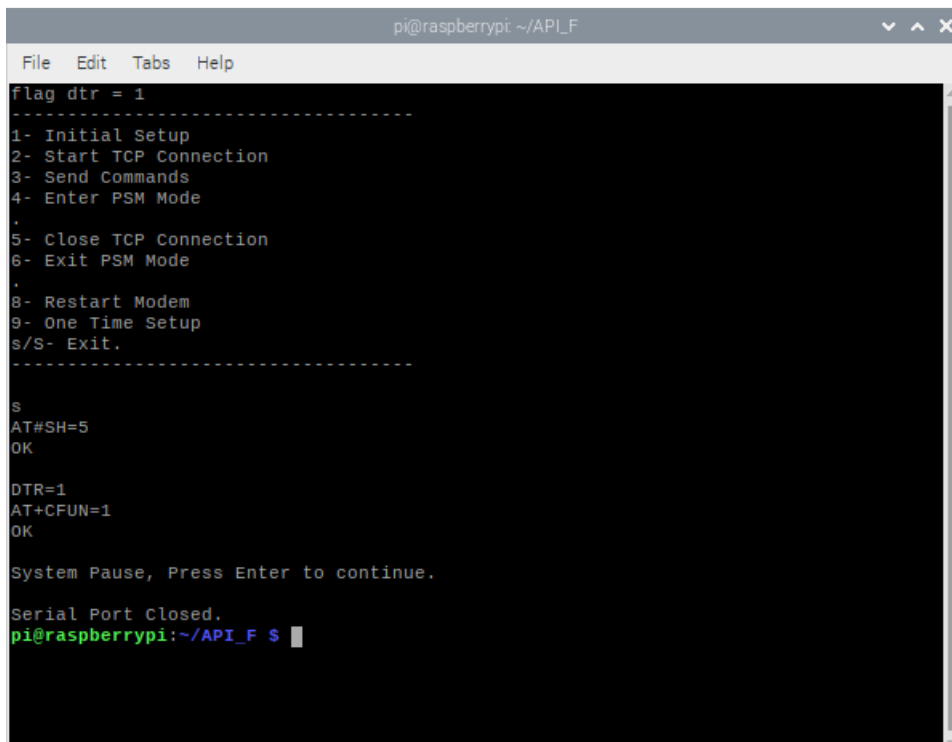
- Fechar ligação TCP 4.5.



```
pi@raspberrypi: ~/API_F
File Edit Tabs Help
AT#SH=5
OK
AT#SGACT=2,0
OK
System Pause, Press Enter to continue.
System Pause, Press Enter to continue.
```

Figura 4.5: Fechar ligação TCP API

- Desligar API 4.6.



```
pi@raspberrypi: ~/API_F
File Edit Tabs Help
flag dtr = 1
-----
1- Initial Setup
2- Start TCP Connection
3- Send Commands
4- Enter PSM Mode
.
5- Close TCP Connection
6- Exit PSM Mode
.
8- Restart Modem
9- One Time Setup
s/S- Exit.
-----

s
AT#SH=5
OK

DTR=1
AT+CFUN=1
OK

System Pause, Press Enter to continue.

Serial Port Closed.
pi@raspberrypi:~/API_F $
```

Figura 4.6: Desligar API

Capítulo 5

Conclusões

Após a conclusão do projeto, é importante realçar os pontos positivos e negativos ao longo da realização do mesmo.

Como pontos positivos, considera-se a consolidação dos conhecimentos tanto na linguagem C e no ambiente Linux, juntamente com a aprendizagem e consolidação dos protocolos de comunicação da porta série. Com a realização do projeto, foi possível entrar em contacto com *Hardware e Software*, fora do contexto académico, o que apelou bastante à autonomia e a estabelecer contacto com fornecedores.

Como pontos negativos, enumeram-se algumas dificuldades a ultrapassar ao longo da realização dos testes nos modems, devido à falta de informação nos *datasheets*, os problemas da medição dos consumos no modem NB-IOT e as várias barreiras a ultrapassar durante o desenvolvimento da API.

Todos os ficheiros os ficheiros, código, excel e figuras estão disponíveis no *Github* [21].

Referências

- [1] M. Brain, “How modems work.” Available at <https://computer.howstuffworks.com/modem.htm>, April 2000. (Último acesso em 12/06/2021). [Citado nas páginas vii e 5]
- [2] Fibocom, “What do you know about lte cat 1 in iot.” Available at https://www.fibocom.com/en/Blog/info_itemid_2125.html, 2021. (Último acesso em 12/06/2021). [Citado nas páginas vii e 6]
- [3] Telit, “Le910cx hardware user guide.” Available at https://www.telit.com/wp-content/uploads/2017/11/Telit_LE910C1_Hardware_User_Guide_r1.12.pdf. [Citado nas páginas vii e 7]
- [4] Quectel, “Quectel bc66.” Available at <https://www.elecomes.com/collections/quectel-development-kit/products/bc66nbteb-kit-quectel-bc66-lte-nbiot-test-and-development-board?variant=30605235814533>. (Último acesso em 13/06/2021). [Citado nas páginas vii e 8]
- [5] B. tecnho guys, “Power saving mode.” Available at <https://www.blacktechnoguys.com/2018/12/power-saving-mode-psm-part-ii.html>, December 2018. (Último acesso em 17/06/2021). [Citado nas páginas vii e 24]
- [6] A. Upale, “Lte cat m1 vs nb-iot vs lora.” Available at <https://www.semiconductorstore.com/blog/2018/LTE-Cat-M1-vs-NB-IoT-vs-LoRa-Comparing-LPWANs-Symmetry-Blog/3496/>, September 2018. (Último acesso em 17/06/2021). [Citado nas páginas vii e 25]
- [7] Speedcheck, “Modem.” Available at <http://www.speedcheck.org/pt/wiki/modem/>, 2008. (Último acesso em 12/06/2021). [Citado na página 5]
- [8] R. Kayne, “What is a modem?.” Available at www.easytechjunkie.com/what-is-a-modem.htm, 2007. (Último acesso em 12/06/2021). [Citado na página 5]
- [9] u blox, “Lte cat 1.” Available at u-blox.com/en/technologies/lte-cat-1, 2020. (Último acesso em 12/06/2021). [Citado na página 6]

-
- [10] D. Wolbert, “Nb-iot and cat-m1 vs. cat-1.” Available at <https://www.hologram.io/blog/nb-iot-vs-cat-m1-vs-cat-1>, 2021. (Último acesso em 13/06/2021). [Citado nas páginas 6 e 8]
- [11] G. Vos, “What is narrowband iot (nb-iot)?.” Available at <https://www.sierrawireless.com/iot-blog/what-is-nb-iot/>, 2021. (Último acesso em 13/06/2021). [Citado na página 8]
- [12] Telit, “Telit at controller.” Available at https://www.telit.com/download-file/telit_at_controller_3-5-4_xfp_4-2-2/. (Último acesso em 13/06/2021). [Citado na página 11]
- [13] E. Team, “Serial port monitor.” Available at <https://www.com-port-monitoring.com/>. (Último acesso em 13/06/2021). [Citado na página 12]
- [14] H. group, “Hercules server.” Available at <https://www.hw-group.com/software/hercules-setup-utility>. (Último acesso em 13/06/2021). [Citado na página 15]
- [15] Docklight, “Docklight.” Available at <https://docklight.de/downloads/>. (Último acesso em 13/06/2021). [Citado na página 18]
- [16] Telit, “Telit evb hardware user guide r3.” Available at https://www.telit.com/wp-content/uploads/2020/04/Telit_EVB_Hardware_User_GUide_r3.pdf. [Citado na página 21]
- [17] Quectel, “Bc66-na&bc66 difference introduction.” Available at https://www.quectel.com/download/quectel_bc66-nabc66_difference_introduction_v1-0/. (Último acesso em 17/06/2021). [Citado na página 23]
- [18] Quectel, “Bc66&bc66-naat commands manual.” Available at https://www.quectel.com/UploadImage/Downlad/Quectel_BC66&BC66-NA_AT_Commands_Manual_V2.0.pdf. (Último acesso em 17/06/2021). [Citado na página 23]
- [19] Thales, “Psm calculator.” Available at <https://www.thalesgroup.com/en/markets/digital-identity-and-security/iot/resources/developers/psm-calculation-tool>. (Último acesso em 17/06/2021). [Citado na página 24]
- [20] G. Vos, “What is edrx (extended discontinuous reception)?.” Available at <https://www.sierrawireless.com/iot-blog/edrx-lpwa/>, June 2020. (Último acesso em 17/06/2021). [Citado na página 25]

-
- [21] J. Silva, “Ficheiros projeto.” Available at https://github.com/JoseSilva00/LEEC_PESTA. [Citado na página 43]

Anexo A

Código Integral

A.1 Main.c

```
1  /**
2   * @file main.c
3   * Open serial port, send AT commands and receive modem response.
4   */
5   #include "main.h"
6   /**
7   * Opens Serial port and sets the correct settings
8   * Error on opening serial Port, returns -1
9   * Settings:
10  * No parity
11  * 1 stop bits
12  * 8 byte size
13  * Baudrate:115200
14  *
15  */
16  int OpenSerial()
17  {
18      int serial_port = open("/dev/ttyUSB0", O_RDWR);
19
20      if (serial_port == -1) /* Error Checking */
21      {
22          #if DEBUG == 1
23              printf("\n Error! in Opening ttyUSB0  ");
```

```

24 #endif
25     return -1;
26 }
27 else
28 {
29 #if DEBUG == 1
30     printf("\n    ttyUSB0 Opened Successfully ");
31 #endif
32 }
33 // Create new termios struc, we call it 'tty' for convention
34 struct termios tty;
35
36 // Read in existing settings, and handle any error
37 if (tcgetattr(serial_port, &tty) != 0)
38 {
39 #if DEBUG == 1
40     printf("Error %i from tcgetattr: %s\n", errno, strerror(
41         errno));
42 #endif
43     return -1;
44 }
45
46 tty.c_cflag &= ~PARENB; // Clear parity bit, disabling parity
47 tty.c_cflag &= ~CSTOPB; // Clear stop field, only one stop bit
48                          // used in communication
49 tty.c_cflag &= ~CSIZE; // Clear all bits that set the data
50                          // size
51 tty.c_cflag |= CS8; // 8 bits per byte
52 //tty.c_cflag |= CRTSCTS; // Disable RTS/CTS hardware
53                          // flow control
54 tty.c_cflag |= CREAD | CLOCAL; // Turn on READ & ignore ctrl
55                          // lines (CLOCAL = 1)
56
57 tty.c_lflag &= ~ICANON;
58 tty.c_lflag &= ~ECHO; // Disable echo
59 tty.c_lflag &= ~ECHOE; // Disable erasure
60 tty.c_lflag &= ~ECHONL; // Disable new-line
61                          // echo
62 tty.c_lflag &= ~ISIG; // Disable
63                          // interpretation of INTR, QUIT and SUSP
64 tty.c_iflag &= ~(IXON | IXOFF | IXANY); // Turn off s/w flow
65                          // ctrl
66 tty.c_iflag &= ~(IGNBRK | BRKINT | PARMRK | ISTRIP | INLCR |
67     IGNCR | ICRNL); // Disable any special handling of received
68     bytes
69
70 tty.c_oflag &= ~OPOST; // Prevent special interpretation of
71     output bytes (e.g. newline chars)

```

```

61     tty.c_oflag &= ~ONLCR; // Prevent conversion of newline to
        carriage return/line feed
62
63     tty.c_cc[VTIME] = 10; // Wait for up to 1s (10 deciseconds),
        returning as soon as any data is received.
64     tty.c_cc[VMIN] = 0;
65
66     // Set in/out baud rate to be 115200
67     cfsetispeed(&tty, B115200);
68     cfsetospeed(&tty, B115200);
69
70     // Save tty settings, also checking for error
71     if (tcsetattr(serial_port, TCSANOW, &tty) != 0)
72     {
73 #if DEBUG == 1
74         printf("Error %i from tcsetattr: %s\n", errno, strerror(
            errno));
75 #endif
76         return -1;
77     }
78
79     return serial_port;
80 }
81 ///////////////////////////////////////////////////
82 /**
83  * Set of commands to be run on startup of the modem
84  *   Enable error report
85  *   Set RI line pulse duration to 500ms
86  *   Unlocks the sim card
87  */
88 void InitialSetup()
89 {
90     char msg[5][50];
91     strcpy(msg[0], AT);
92     strcpy(msg[1], CFUN "1\r"); //PSM OFF
93     strcpy(msg[2], CMEE); //Enables the report of the
        error result code in verbose format.
94     strcpy(msg[3], PSMRI); //Enables the ring indicator
        line (RI) response to an event while the modem is in power
        saving mode. Sets the pulse duration to 500ms.
95     strcpy(msg[4], CPIN PIN "\r"); //SIM card PIN =0000
96
97     for (int i = 0; i < 5; i++)
98         ReadWrite(msg[i]);
99 }
100 ///////////////////////////////////////////////////
101 /**
102  *   Start a TCP connection
103  *   Opens context 2

```

```

104 *   Opens the modem socket N 5 and connects to a TCP server
105 */
106 void OpenSocketServer()
107 {
108     char msg[2][50];
109
110     strcpy(msg[0], SGACT SGACT_CONTEXT ",1\r");
111     strcpy(msg[1], SD Socket_TCP ",0," PORT ",\" IPadd "\",0,0,1\r");
112
113     ReadWrite(msg[0]);
114     ReadWrite(msg[1]);
115
116     SendCommand();
117 }
118 //////////////////////////////////////
119
120 /**
121 *   Terminates the TCP connection by closing the socket and
122     deactivating the context
123 */
124 void CloseSocketServer()
125 {
126     char msg[2][50];
127
128     strcpy(msg[0], SH Socket_TCP "\r");
129     strcpy(msg[1], SGACT SGACT_CONTEXT ",0\r");
130
131     ReadWrite(msg[0]);
132     ReadWrite(msg[1]);
133 }
134 //////////////////////////////////////
135
136 /**
137 * To send data to the modem we have to write in the file ttyUSB0
138 */
139 void SendData(char msg[])
140 {
141     write(serial_port, msg, strlen(msg));
142 }
143 //////////////////////////////////////
144 /**
145 * To read the modem response, read the file ttyUSB0
146 */
147 void ReadData()
148 {
149     char read_buf[256];
150
151     // set everything to 0 so we can

```



```

151     // call printf() easily.
152     memset(&read_buf, '\0', sizeof(read_buf));
153
154     // Read bytes.
155     int num_bytes = read(serial_port, &read_buf, sizeof(read_buf))
        ; ///get modem response
156
157     #if DEBUG == 1
158         printf("%s\n", read_buf); ///print modem response
159     #endif
160     fflush(stdin);
161     fflush(stdout);
162 }
163 ///////////////////////////////////////////////////
164
165 /**
166  * set the Data Terminal Ready(DTR) to 1 to leave Power Save Mode(
        PSM) and communicate with the modem
167  */
168 void SetDTR()
169 {
170     int status;
171     ioctl(serial_port, TIOCMGET, &status); //GET DTR PIN status
172
173     status |= TIOCM_DTR;
174     ioctl(serial_port, TIOCMSET, status); //SET DTR PIN to 1 if to
        communicate, if the modem is in power save
175     #if DEBUG == 1
176         printf("\nDTR=1\n");
177     #endif
178 }
179 ///////////////////////////////////////////////////
180
181 /**
182  * set the DTR to 0 to enter PSM
183  */
184 void ClearDTR()
185 {
186     int status;
187     ioctl(serial_port, TIOCMGET, &status); //GET DTR PIN status
188
189     status &= ~TIOCM_DTR;
190     ioctl(serial_port, TIOCMSET, status); //SET DTR PIN to 0, if
        the modem is in power save mode
191     #if DEBUG == 1
192         printf("\nDTR=0\n");
193     #endif
194 }
195 ///////////////////////////////////////////////////

```

```

196
197 /**
198  * Checks the at command sent:
199  * if AT+CFUN=5, then sets the DTR line to 0 to enter PSM
200  * if AT+CFUN=1, then sets the DTR line to 1 to leave PSM
201  */
202 void CheckCFUN(char msg[])
203 {
204     if (strcmp(msg, "at+cfun=5\r") == 0 || strcmp(msg, "AT+CFUN=5\r") == 0)
205     {
206         FLAG_DTR = 0;
207         ClearDTR();
208     }
209
210     if (strcmp(msg, "at+cfun=1\r") == 0 || strcmp(msg, "AT+CFUN=1\r") == 0)
211     {
212         FLAG_DTR = 1;
213         SetDTR();
214     }
215 }
216 //////////////////////////////////////
217 /**
218  * Checks if the FLAG_DTR
219  * @details If FLAG_DTR==0, the modem is in PSM, Sets the DTR line
220  *          to 1
221  *          Sends Data
222  *          Receives the modem Response
223  *          If FLAG_DTR==0, the modem must turn back to PSM, Sets
224  *          the DTR line to 0
225  */
226 void ReadWrite(char msg[])
227 {
228     if (FLAG_DTR == 0)
229     {
230         SetDTR();
231     }
232
233     CheckCFUN(msg);
234     SendData(msg);
235     ReadData(msg);
236
237     if (FLAG_DTR == 0)
238     {
239         ClearDTR();
240     }
241 }
242 //////////////////////////////////////

```

```

241  /**
242   * Scans user inputted commands and sends to modem
243   */
244  void SendCommand()
245  {
246      char msg[80];
247      #if DEBUG == 1
248          printf("\nGo Back to Menu Send: back");
249      #endif
250      while (strcmp(msg, "back\r") != 0 && strcmp(msg, "BACK\r") !=
251              0)
252      {
253          printf("\nCODE:");
254          scanf("%s", msg);
255          strncat(msg, "\r", 2);
256          ReadWrite(msg);
257      }
258      //////////////////////////////////////
259  /**
260   * Run only the first time the modem is started.
261   * Socket Configuration commands.
262   * Selects PDP context identifier 2, selects "IP" data protocol,
263     sets "internet" APN.
264   * Selects the socket 5, sets SRING URC mode to display the
265     connId and the amount of data received, disables TCP keepalive
266     timer timeout, sets the socket initial connection timeout to 60
267     s, disables the max packet timeout(not used in command mode).
268   * Selects the socket 5, enables the indication of connId socket
269     and closure cause when a NO CARRIER indication occurs.
270   * Selects the socket 5, the PDP context 2, sets the max packet
271     size for the TCP/IP stack to 1500 (not used in command mode),
272     disables the socket exchange inactivity timeout, sets the
273     socket initial connection timeout to 60s, disables the max
274     packet timeout (not used in command mode).
275   */
276  void OneTime()
277  {
278      char msg[10][50];
279
280      strcpy(msg[0], CGDCONT Context " , \"IP\" , \" \" APN \" \",
281              \"\",0,0,0,0\r");
282      strcpy(msg[1], SCFGEXT Socket_TCP SCFGEXT_N);
283      strcpy(msg[2], SCFGEXT2 Socket_TCP SCFGEXT2_N);
284      strcpy(msg[3], SCFG Socket_TCP SCFG_N);
285
286      for (int i = 0; i < 4; i++)
287      {
288          ReadWrite(msg[i]);

```

```

279     }
280 }
281 //////////////////////////////////////////////////
282 /**
283  * Reboot Modem instantly:AT#ENHRST=1,0
284  *
285  */
286 void Restart()
287 {
288     char msg[50];
289     strcpy(msg, Reboot); //Restart Modem
290     #if DEBUG == 1
291         printf("\nRestarting...\n");
292     #endif
293     ReadWrite(msg);
294 }
295 //////////////////////////////////////////////////
296
297 /**
298  * Enter PSM Mode: AT+CFUN=5
299  */
300 void EnterPSM()
301 {
302     char msg[50];
303
304     strcpy(msg, CFUN5);
305     ReadWrite(msg);
306 }
307 //////////////////////////////////////////////////
308
309 /**
310  * Exit PSM Mode: AT+CFUN=1
311  */
312 void ExitPSM()
313 {
314     char msg[50];
315
316     strcpy(msg, CFUN1);
317     ReadWrite(msg);
318 }
319 //////////////////////////////////////////////////
320 /// Menu of options to interact with the user
321 char menu()
322 {
323     char choice = ' ';
324     do
325     {
326         system("clear");
327         printf("flag dtr = %d\n", FLAG_DTR);

```

```

328     printf("-----\n");
329     printf("1- Initial Setup\n");           //activates error
        report and ri response time 500ms
330     printf("2- Start TCP Connection\n"); //activates context
        and starts a TCP connection
331     printf("3- Send Commands\n");           //user freely sends
        commands
332     printf("4- Enter PSM Mode\n");
333     printf(".\n");
334     printf("5- Close TCP Connection\n");
335     printf("6- Exit PSM Mode\n");
336     printf(".\n");
337     printf("8- Restart Modem\n"); //turn the modem off or
        restart
338     printf("9- One Time Setup\n"); //defines the settings for
        the socket 5 and turns the leds off
339     printf("s/S- Exit.\n");
340     printf("-----\n\n");
341     scanf("%c", &choice);
342 } while (choice != 's' && choice != 'S' && choice != '1' &&
        choice != '2' && choice != '3' && choice != '4' && choice
        != '5' && choice != '6' && choice != '7' && choice != '8'
        && choice != '9');
343
344     return choice;
345 }
346 //////////////////////////////////////////
347 int main()
348 {
349     system("clear");
350
351     char choice = ' ';
352
353     serial_port = OpenSerial();
354
355     if (serial_port == -1)
356     {
357
358     #if DEBUG == 1
359         printf("Error In Serial Port");
360     #endif
361         return -1;
362     }
363     #if DEBUG == 1
364         SystemPause();
365     #endif
366
367     pid_t pid = fork();
368

```

```
369     if (pid== 0)
370     {
371         while(1)
372         {
373             ReadData();
374         }
375     }
376
377
378
379     do
380     {
381         choice = menu();
382
383         switch (choice)
384         {
385             case '1':
386             {
387                 system("clear");
388
389                 InitialSetup();
390 #if DEBUG == 1
391                 SystemPause();
392                 SystemPause();
393 #endif
394                 break;
395             }
396
397             case '2':
398             {
399                 system("clear");
400
401                 OpenSocketServer();
402 #if DEBUG == 1
403                 SystemPause();
404                 SystemPause();
405 #endif
406                 break;
407             }
408
409             case '3':
410             {
411                 system("clear");
412                 SendCommand(); //Write commands
413 #if DEBUG == 1
414                 SystemPause();
415                 SystemPause();
416 #endif
417                 break;
```

```
418         }
419
420         case '4':
421         {
422             system("clear");
423
424             EnterPSM();
425 #if DEBUG == 1
426             SystemPause();
427             SystemPause();
428 #endif
429             break;
430         }
431
432         case '5':
433         {
434             system("clear");
435
436             CloseSocketServer();
437 #if DEBUG == 1
438             SystemPause();
439             SystemPause();
440 #endif
441             break;
442         }
443
444         case '6':
445         {
446             system("clear");
447
448             ExitPSM();
449 #if DEBUG == 1
450             SystemPause();
451             SystemPause();
452 #endif
453             break;
454         }
455
456         case '8':
457         {
458             system("clear");
459             Restart(); //Reboot Modem
460 #if DEBUG == 1
461             SystemPause();
462             SystemPause();
463 #endif
464             break;
465         }
466         case '9':
```

```
467         {
468             system("clear");
469             OneTime();
470             SystemPause();
471             break;
472         }
473     }
474 } while (choice != 's' && choice != 'S');
475
476 CloseSocketServer();
477 ExitPSM();
478 SystemPause();
479 close(serial_port);
480 kill(pid, SIGTERM);
481
482 #if DEBUG == 1
483     printf("\nSerial Port Closed.\n");
484 #endif
485     return 0;
486 }
487
488 void SystemPause()
489 {
490     printf("\nSystem Pause, Press Enter to continue.\n");
491     int c = getchar();
492     c++;
493 }
```

Listagem A.1: Main

A.2 Main.h

```

1  /**
2   * @file main.h
3   */
4
5  // C library headers
6  #include <stdio.h>
7  #include <string.h>
8  #include <stdlib.h>
9
10 // Linux headers
11 #include <fcntl.h>      // Contains file controls like O_RDWR
12 #include <errno.h>      // Error integer and strerror() function
13 #include <termios.h>    // Contains POSIX terminal control
14                          // definitions
15 #include <unistd.h>      // write(), read(), close()
16 #include <sys/ioctl.h>   // Control dtr line
17 #include <signal.h>
18
19 #define DEBUG 1 // 1 -> Show errors and modem response
20
21
22 int serial_port;      //Serial Port
23 int FLAG_DTR;         //FLAG==1, dtr is on and modem is free to
24                       //communicate
25                       //FLAG==0, Modem is in PSM
26
27
28 int OpenSerial();
29 void InitialSetup();
30 void OpenSocketServer();
31 void CloseSocketServer();
32 void SendData(char *);
33 void ReadData();
34 void SetDTR();
35 void ClearDTR();
36 void CheckCFUN(char *);
37 void ReadWrite(char *);
38 void SendCommand();
39 void Onetime();
40 void Restart();
41 void EnterPSM();
42 void ExitPSM();
43 char menu();
44 void SystemPause();

```

```

45 #define APN "internet"          ///< ISP APN to use. Altice ->
    internet / Vodafone ->internet.vodafone.pt
46 #define IPadd "161.230.159.26" ///< IP address to connect to
    server
47 #define PORT "5000"             ///< Server PORT
48 #define PIN "0000"             ///< SIM card PIN
49
50
51 //Restart
52 #define Reboot "AT#ENHRST=1,0\r"
53
54 //Initial Setup
55 #define AT "AT\r"
56 #define CMEE "AT+CMEE=2\r"      ///< error report
57 #define CPIN "AT+CPIN="        ///< SIM card pin
58 #define PSMRI "AT#PSMRI=500\r" ///<
59 #define CFUN "AT+CFUN="        ///< PSM mode
60 #define CFUN5 "AT+CFUN=5\r"    ///< Turn on PSM
61 #define CFUN1 "AT+CFUN=1\r"    ///< Turn off PSM
62
63 //SocketServer
64 #define SGACT "AT#SGACT="       ///< Context management
65 #define SGACT_CONTEXT "2"
66 #define SD "AT#SD="            ///< Start TCP Connection
67 #define SH "AT#SH="            ///< Close TCP connection
68 #define Socket_TCP "5"        ///< Modem socket to open TCP
    connection
69
70 //OneTime                      ///< Socket management
71 #define CGDCONT "AT+CGDCONT="
72 #define Context "2"
73 #define SCFGEXT "AT#SCFGEXT="
74 #define SCFGEXT_N ",1,0,0,0,0\r"
75 #define SCFGEXT2 "AT#SCFGEXT2="
76 #define SCFGEXT2_N ",0,0,0,0,2\r"
77 #define SCFG "AT#SCFG="
78 #define SCFG_N ",2,1500,0,600,0\r"

```

Listagem A.2: Header

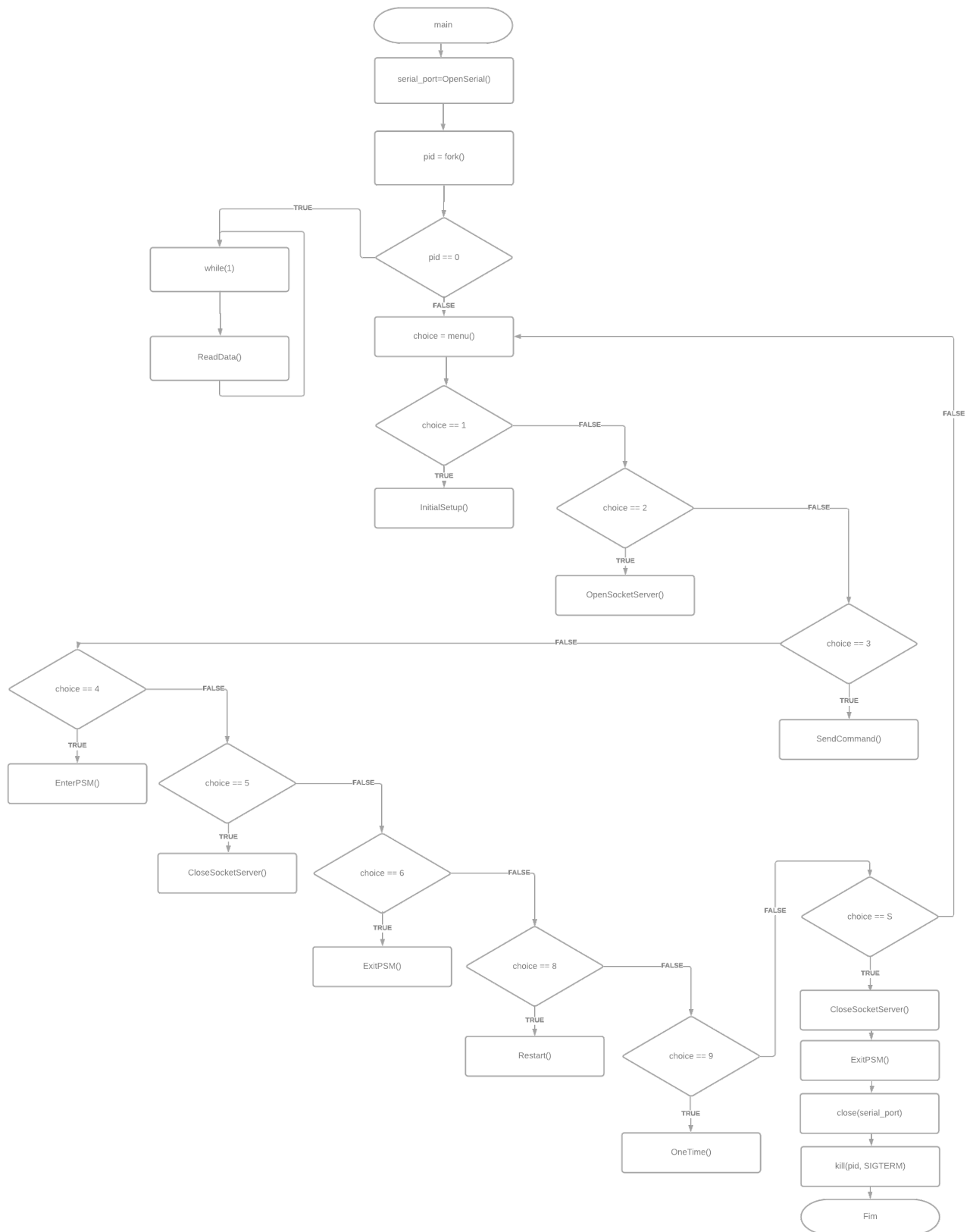


Figura A.1: Fluxograma base API