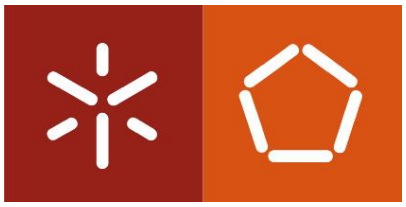
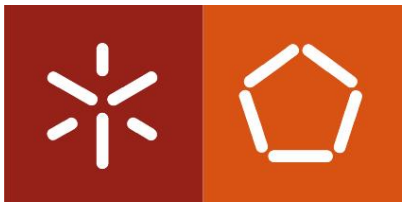


Universidade do Minho
Escola de Engenharia

João Manuel Costa Enes da Lage

Wi-Fi Tag – Módulo de *software* para aquisição e envio de *fingerprints* baseado no sistema de localização *indoor* RADAR

Novembro de 2014



Universidade do Minho
Escola de Engenharia

João Manuel Costa Enes da Lage

Wi-Fi Tag – Módulo de *software* para aquisição e envio de *fingerprints* baseado no sistema de localização *indoor* RADAR

Dissertação de Mestrado
Mestrado Integrado em Engenharia Eletrónica Industrial e Computadores

Trabalho efetuado sob a orientação do:
Professor Doutor Jorge Cabral

Novembro de 2014

AGRADECIMENTOS

Esta dissertação simboliza o meu reconhecimento académico como mestre na área que estudo. Que tristeza me dá saber que este pequeno texto não deve ultrapassar as quinhentas palavras, poderia fazê-lo do tamanho do resto do documento! Enfim, não é só o reconhecimento académico que esta tese simboliza mas também um esforço inalcançável de um conjunto de pessoas que fizeram com que eu conseguisse chegar onde cheguei, aqui! Estou muitíssimo feliz e não posso esquecer o apoio incansável da minha família, que sem ela não conseguiria ter este momento de reflexão e de retrospetiva de tudo o que me passou nestes últimos anos académicos. Sem este apoio económico e moral inigualável consigo afirmar que não conseguiria ter melhor auxilio do que aquele que me foi concedido. Agradeço incondicionalmente ao meu pai José, mãe Ana, irmão Carlos Diogo "tok" e aos meus avós Áurea, João, Madalena e Manuel por tudo o que me deram, sempre de mão aberta para me amparar.

Não posso também esquecer o apoio dado pelos amigos que conheci na universidade que sempre me ajudaram a ultrapassar as maiores adversidades e também me fizeram dar as maiores gargalhadas nos serões que fazíamos juntos. Não quero nomear ninguém mas sinto que ia acabar por parecer muito impessoal ao deixar de lado os nomes Davide "Cuty" Guimarães, Emanuel "Futsal" Costa, Eurico "Monção" Moreira, Fábio "Areal" Ferreira, João "Phelps" Sousa, Joel "Coives" Dias, Luís "Lampreia" Costa, Pedro "Seixo" Pereira, Pedro "Mirones" Ribeiro, Pedro "Penafiel" Rocha e Tiago "Castor" Fernandes. Outra pessoa que também não posso deixar de lado e que, apesar de não estar incluída no mesmo meio académico, me apoiou imenso durante todos estes anos foi a Margarida Rosário, um agradecimento especial para ti!

Reconheço especialmente o colega e amigo Tiago Fernandes com quem tive o prazer de trabalhar no decorrer do desenvolvimento desta dissertação. Bons momentos vivemos nestes últimos anos e principalmente nesta reta final onde tudo se intensificou, fazendo com que a entreajuda e espirito de equipa crescessem sem igual.

Quero também agradecer por todo o apoio e oportunidade concedida pelo meu orientador Jorge Cabral, que me guiou neste projeto, como também o João Brito que geriu e proporcionou melhores soluções de implementação para que trabalho alcançasse sucesso.

Finalmente quero formalizar um agradecimento ao ESRG, colegas do CCG, colegas e amigos do curso de MIEEIC, técnicos do DEI e à excelente academia que me acolheu durante estes cinco anos.

RESUMO

Palavras-chave: IEEE 802.11 ; localização *wireless* ; RSSI ; *fingerprints* ; mapa rádio ; NLoS ; indicador de potência de sinal recebida ; Wi-Fi

Muitas técnicas foram propostas e desenvolvidas para localização *outdoor* sendo estas ideias num contexto LoS (*Line-of-Sight*). Com o sucesso da tecnologia GPS e com as constantes exigências por parte dos utilizadores para ambientes inseridos em contextos NLoS (*Non Line-of-Sight*) muitas tecnologias foram propostas para combater esta lacuna de localização *indoor*. Este documento visa a integração da tecnologia IEEE 802.11, empregando uma técnica que utiliza o indicador de potência de sinal RSSI (*Received Signal Strenght Indicator)* e da abordagem de recolha por *fingerprints*. Acompanhado com este sistema existe um acelerómetro, inserido numa máquina de estados, responsável por iniciar as execuções de leitura e posterior envio das *fingerprints*.

Os principais objetivos desta dissertação visam a aquisição e envio destas *fingerprints* encapsuladas por uma notação de objetos para posterior monitorização e estimativa de localização.

Relativamente aos resultados, estes mostram um *feedback* bastante positivo e é de relevar o facto de todos os objetivos terem sido concluídos com sucesso durante o desenvolvimento desta dissertação.

ABSTRACT

Keywords: IEEE 802.11 ; wireless location ; RSSI ; fingerprints ; radio map ; NLoS ; received signal strength indicator ; Wi-Fi

Many techniques in the context of LoS (Line-of-Sight) were proposed and developed for the outdoor location. With the success of the GPS technology and the constant demands by the user for NLoS (Non Line-of-Sight) embedded environments, several technologies were suggested to tackle this indoor location gap. This document aims to portray the integration of the IEEE 802.11 technology, applying a technique that utilizes the RSSI (Received Signal Strength Indicator) signal strength indicator as well as a fingerprints assembly approach. Tagging along this system, and inserted in a state machine, exists an accelerometer responsible for initiating the reading orders and, later, sending the fingerprints.

The main objectives of this dissertation aim the data acquisition and sending of the encapsulated information by an object notation language to, hereafter, monitor and estimate the user's location.

About the obtained results, those show a really positive feedback, needing to be highlighted the fact of all the objectives were achieved with success during the dissertation development.

ÍNDICE GERAL

ABREVIATURAS E SIGLAS

A – Ampere

ACK – Acknowledge

ADC – Analog-to-digital converter

AoA – Angle of arrival

AP – Access Point

API – Application Programming Interface

ARQ – Automatic Repeat reQuest

bps – bits por segundo

BPSK – Binary Phase Shift Keying

(B)SSID – (Basic) Service Set Identification

CCK – Complementary Code Keying

CLK – Clock

CMSIS – Cortex Microcontroller Software Interface Standard

CRC – Cyclic Redundancy Check

CS – Chip Select / SS – Slave Select

CSMA/CA – Carrier Sense Multiple Access with Collision Avoidance

CSMA/CD – Carrier Sense Multiple Access with Collision Detection

DC – Direct Current

DCF – Distributed Coordinated Function

DLL – Data Link Layer

DNS – Domain Name System

DSSS – Direct Sequence Spread Spectrum

(D)VCS – (Distributed) Version Control System

ESRG – Embedded System Research Group

FCS – Frame check sequence

FHSS – Frequency-Hopping Spread Spectrum

FTP – File Transfer Protocol

G – aceleração gravitacional (força g) 1G = aceleração sentida na terra ao nível da água do mar

GB – Gigabyte

GPIO – General Purpose Input/Output

GPS – Global Positioning System (GPS)

GUI – Graphical User Interface

HAL – Hardware Abstraction Layer

HDK – Hardware Development Kit

HTTP – Hypertext Transfer Protocol

HTTPs – Hypertext Transfer Protocol secure

Hz – Hertz (Frequência)

I2C – Inter-Integrated Circuit

IDE – Integrated Development Environment

IEEE – Institute of electrical and electronics engineers

IP – Internet Protocol

JSON – JavaScript Object Notation

kB – kilobyte

kNN – k-Nearest Neighbors

LAN – Local Area Network

LED – Light Emitting Diode

LLC – Logical Link Control

LoS – Line of Sight

LSB – Location Based System

MB – megabyte

MAC – Media Access Control

MCU – Microcontroller Unit

mDNS – multicast Domain Name System

MIMO – multiple-input multiple-output

MISO – Master input, slave output

MOSI – Master output, slave input

MSDU – MAC Service Data Unit

NLoS – Non Line-of-Sight

OFDM – Orthogonal Frequency-Division Multiplexing

OSI – Open Systems Interconnection

PCF – Point Coordinated Function

PEAP – Protected Extensible Authentication Protocol

QAM – Quadrature Amplitude Modulation

QPSK – Quadrature Phase Shift Keying

RAM – Random access memory

RFID – Radio-Frequency IDentification

ROM – Read Only Memory

RSS(I) – Received Signal Strength (Indicator)

RTOS – Real Time Operating System

s – Segundo (Tempo)

SDK – Software Development Kit

SFD – Start Frame Delimiter

SHA – Secure Hash Algorithm

SoC – System on Chip

SPI – Serial peripheral interface

SRAM – Static Random-Access Memory

TCP – Transmission Control Protocol

T(D)oA – Time (Difference) Of Arrival

TI – Texas Instruments

UDP – User Datagram Protocol

UM – Universidade do Minho

U(S)ART – Universal (Synchronous/)Asynchronous Receiver/Transmitter

USB – Universal serial bus

V – Volt (Tensão)

VoIP – Voice over Internet Protocol

VSP – Virtual Serial Port

WEP – Wired Equivalent Privacy

Wi-Fi – Wireless Fidelity

WLAN – Wireless Local Area Network

WPA – Wi-Fi Protected Access

ÍNDICE DE FIGURAS

Índice de Figuras

ÍNDICE DE TABELAS

Capítulo 1 Introdução

Com o aparecimento dos sistemas embebidos iniciada em grande quantidade nos anos 60, foi então gerada até hoje uma colossal revolução no que toca ao desenvolvimento deste tipo de sistemas. Estes sistemas, cada vez mais dinâmicos, preemptivos, móveis e autossustentáveis conferem ao utilizador uma maior confiança e segurança nas mais variadas atividades diárias. Todas estas capacidades presentes neste tipo de sistemas foram adquiridas com o avanço tecnológico inevitável, procura e necessidade dos seus utilizadores em assegurar e cobrir o meio ambiente e todas as pessoas que o rodeiam.

Com isto, ao desenvolver sistemas deste género, deveremos ceder ao sistema a capacidade de operar nos mais variados cenários onde o correto funcionamento deste é vital tendo em conta a segurança do público e do meio. Esta capacidade é adquirida quando todas as adversidades que se poderão suceder são integradas e previstas em casos de uso para que quando aconteçam o sistema saiba lidar e responder de uma forma mais rápida, correta e segura tendo em conta todas as variáveis. "from anti-lock braking systems in automobiles, to fly-by-wire aircraft, to shut-down systems at nuclear power plants. It is, therefore, vital that engineers are aware of the safety implications of the systems they develop." [1]

1.1 Comunicação sem-fios em sistemas embebidos

Com a aparição dos sistemas embebidos a necessidade de aliá-los a outros módulos criados foi infinda, fazendo com que toda esta comunicação microcontrolador-módulo a fosse *hard wired*. Após este avanço o sector necessitou de criar e centralizar sistemas mais complexos onde a necessidade incidia sobre a mobilidade, onde deixaríamos de ter sistemas estáticos e toda a sua comunicação entre outros dispositivos seria através de ondas rádio.

Posto isto, ganharíamos acesso e comunicação remotos, podendo monitorizar sistemas distribuídos com mais complexidade.

"The need of wireless communication systems has revealed an exponential growth in the last few years, having as advantages its mobility, low-powered modules and microcontrollers, not wired environments and accessibility removing any node from the system without any disruption of its remainder." [2]

Sendo que a mobilidade de um sistema está sempre dependente do modo como este é energeticamente alimentado, surgiu então sistemas alimentados a baterias. Com este avanço iniciou-se então uma maior preocupação relativa aos consumos energéticos e a uma cuidada escolha relativa de componentes *low-power* para o sistema a construir.

"When taking advantage and mixing the battery powered and wireless embedded systems, those make a very welcoming scenario, especially when the population's needs are higher in respect of energy savings, resulting in a greater up-time." [2]

Com todas estas necessidades aliadas à evolução tecnológica começaram a ser criados produtos que focam a conectividade, eficiência, segurança e produtividade.

1.2 Objetivos

O projeto WifiTag, a ser desenvolvido por duas pessoas com divisão em tarefas distintas, tem como objetivo localizar um dispositivo (*tag*) através da identificação das redes *wireless* com a especificação 802.11 que a rodeiam. Cada dispositivo vai então proceder ao scan das redes que se encontram à sua volta e enviar estes valores para um servidor já conhecido.

Este documento, em particular, foca a aquisição de dados a partir do dispositivo móvel. O comportamento deste dispositivo integra um conjunto de estados distintos bem como a comunicação, aquisição e envio da informação com a ajuda de um módulo externo 802.11.

Figura 1.1 - *Overview* operacional do sistema - campo de atuação

1.2.1 Contextualização

Este projeto enquadra-se num contexto hospitalar onde todas as pessoas a localizar transportam consigo uma *tag*. Cada *tag*, com o seu código único, envia informações para o servidor sobre todas as leituras efetuadas.

Do lado do servidor existe um algoritmo que, com a informação prévia de onde estão localizados os APs nessa unidade hospitalar, consegue estimar uma localização aproximada para esse determinado funcionário. O servidor diferenciando qual o dispositivo que enviou essa dada informação, através do campo que o identifica, consegue então canalizar a informação recebida apenas para essa *tag* e assim adquire a capacidade de receber leituras de mais do que uma tag em simultâneo e mesmo assim dividir a informação recebida por funcionários para posterior monitorização.

Todo este projeto foi idealizado num contexto teórico e este relatório trata o sistema presente na chamada *tag*.

1.3 Estrutura da dissertação

Este relatório está contruído respeitando o *waterfall model*, possuindo uma construção de *software* sequencial onde são respeitadas as fases de Iniciação, Análise, Design, Implementação e Resultados. A estrutura do modelo de desenvolvimento em *waterfall* pode futuramente ajudar a perceber melhor todos os conceitos do projeto e a sua implementação *top-bottom*.

Inicialmente foi realizada uma análise do sistema onde esta contém uma breve descrição do projeto e todos os requisitos do sistema são apresentados para que o resultado final seja o esperado. Deste modo haverá uma evolução contínua no projeto, sem paragens e redefinição de requisitos. Finalmente irá ser demonstrado graficamente os estados que o sistema deve respeitar e uma visão geral do sistema.

Após a análise do sistema foi então realizado o *design* do mesmo que incide sobre o comportamento do sistema, respeitando os requisitos impostos, a placa de desenvolvimento que deu suporte ao software a desenvolver como também os casos de uso deste projeto.

A fase da pesquisa e análise teórica do sistema foi realizada antes da implementação do sistema em si para que não existam dúvidas e a implementação seja o mais otimizada possível. Garantindo umas boas bases teóricas consegue-se ter outro ponto de vista do sistema a desenvolver e qualquer dúvida existente poderá ser resolvida muito mais facilmente. Esta

pesquisa irá focar algumas camadas do modelo OSI, mais propriamente as camadas física, de ligação, rede e transporte. Também foi realizada uma pesquisa sobre sistemas de controlo de versões em *software* e o IDE utilizado.

No que toca à última grande tarefa deste projeto, a implementação, esta começa por testar todo o sistema por módulos, sendo que inicialmente foi testado o módulo Wi-Fi das Texas Instruments com o modelo CC3000 e os seus consumos energéticos. Depois foi testado o envio de dados a partir da *suite* TCP/IP e UDP, baseados nas pesquisas sobre os conceitos teóricos, e as vantagens e desvantagens na escolha do protocolo a usar. Também individualmente foi desenvolvida a máquina de estados responsável por transitar o sistema pelos variados estados como também algumas configurações e testes ao acelerómetro. Finalmente todos os módulos acima referidos foram reunidos, integrados e testados resultando num sistema unitário.

A documentação gerada no decorrer desta dissertação foi guardada num servidor *wiki* cedido pelo Embedded Systems Research Group (ESRG).

Capítulo 2 Estado de arte

Neste segundo capítulo irá ser realizada uma pesquisa e reflexão sobre estudos e sistemas que cumpram o mesmo objetivo ou que relacionem e incluam conhecimentos base relativamente ao projeto descrito neste relatório. É essencial estudar o que já foi investigado e implementado nesta área fazendo com que os objetivos deste projeto sejam mais claros e para que se consiga situar o leitor no contexto do tema.

Posto isto, neste tópico irão ser abordados mais ao pormenor *papers* técnicos onde foram estudadas abordagens, algoritmos e implementações que facilitem e melhorem o desempenho das aplicações referidas.

2.1 Sistemas de posicionamento

Hoje em dia muita atenção é dada aos sistemas de posicionamento onde muitos investigadores declaram como o GPS a solução mais viável para ambientes *outdoor*. Esta tecnologia, com uma precisão sem precedentes, confiança e disponibilidade em ambientes abertos faz com possa ser utilizada em ações militares, defesa nacional, transportação, pesca, mapeamento e dia-a-dia pessoal. [3] Esta inovação parece ser mesmo a solução mais viável no que toca a sistemas de localização em espaço aberto, porém as suas limitações são notáveis quando esta tecnologia é utilizada em grandes cidades, onde as urbanizações são muito densas, e em espaços fechados. Esta enorme desvantagem, relacionada com as crescentes exigências mundiais com sistemas em ambientes NLoS (*Non-Line-of-Sight*) [4], resultou num grande interesse em LSBs (*Location Based Services*) por parte de variados grupos de investigação em localizações indoor. [5]

2.1.1 Metodologias para localização indoor

Nos dias de hoje, grande parte da atividade humana é realizada em espaços fechados, em que nas condições atuais os satélites e navegações dificilmente conseguem manter os requisitos de posicionamento. Existem inúmeras soluções aplicadas em sistemas de posicionamento indoor como por exemplo: acesso à rede móvel, raios infravermelhos, ultrassons, RFID e Bluetooth. Estas tecnologias proporcionam e desempenham um grande papel ao estimar localizações em ambientes NLoS, especialmente quando incluem uma grande densidade urbana. Os métodos de predição em sistemas sem-fios que implementam técnicas de localização são classificados em quatro grandes categorias: Força de Sinal recebido (RSS),

Tempo de chegada (ToA), Diferença de Tempos de Chegada (TDoA) e Ângulo de Chegada (AoA). Os últimos três métodos podem facilmente ser bloqueados e/ou ludibriados por objetos móveis, implicando valores de distâncias pouco precisos ou até errados.[3], [4] ”But the more existing systems the more problems arise.” [5] Posto isto, a criação de novas tecnologias é quase impossível dada ao elevado número de standards e limitações, como também recursos, por exemplo, espectro de frequências para ondas rádio. Em termos de hardware, estas novas tecnologias poderiam requerer um design que seria muito caro e complicado.

2.2 IEEE 802.11 como solução de posicionamento indoor

Uma das tecnologias mais promissoras para sistema de localização em ambientes NLoS incluem aplicações baseadas no *standard* IEEE 802.11. Hoje em dia está presente em todo o lado, e instalado em praticamente todos os dispositivos, hardware que incorpore a tecnologia 802.11. Um facto importante é que esta tecnologia emergiu no mercado para normalizar as comunicações rádio no que toca à troca de informação e acesso à internet a utilizadores móveis. Posto isto, a exponencial procura e conseguinte avanço tecnológico desta invenção fez com que fossem rapidamente investigadas e procuradas soluções onde esta encaixasse o seu principal com algoritmos de localização.[6]

2.2.1 Especificação IEEE 802.11 em sistemas de localização

Posto isto, as vantagens são infindas porque não existe a necessidade de adquirir e instalar esta tecnologia para conseguir estudar e implementar uma solução de localização, visto que, esta está presente e praticamente todo o lado, como referido anteriormente.

Como desvantagens estas são muito distintas visto que existem numerosos fatores que influenciam o comportamento das ondas rádio. Primeiramente, relativamente ao meio que as ondas rádio utilizam para se programar, este tem um papel deveras importante no modo como estas se comporta derivado aos seus próprios fatores. Esses fatores tais como a temperatura, pressão, humidade, interferência, ruído termal podem reduzir, muito significativamente, a precisão da localização estimada derivada à degradação do sinal. [4] O sinal emitido também não é gradualmente atenuado, como também existem outra fatores como a reflexão, refração e absorção por paredes e pessoa, resultando em distorções e *multipath propagations* no recetor. Finalmente, existe também uma grande dependência em termos do *hardware* utilizado para emitir essas ondas rádio. Não está apenas assente a atenuação do sinal em termos do ambiente onde esse sinal é propagado mas também a maneira como é resultada essa propagação. Com

isto, muitos investigadores dependem e realizam pesquisas sobre plataformas comerciais *off-the-shelf* onde a propagação do sinal é diferente de outros produtos, referente às diferentes especificações do produto, por exemplo, gama de operação e ganho. [7], [8]

2.3 Técnicas de localização

Relativamente à abordagem dos sistemas 802.11 este, muitas vezes, depende do RSS (Received Signal Strengh) e com ajuda de algoritmos estima a localização da estação móvel. No ano 2000 foi idealizado um sistema denominado por RADAR [9] que implementa um algoritmo de localização indoor baseado em valores RSS adquiridos e que ainda muitos investigadores o citam para desenvolver LSBs melhorados. Esta técnica comparada com outras, como referido em 2.1.1 , é mais efetiva devido ao efeito *multipath* prevalente em ambientes NLoS que causam ângulos e tempos incorretos. [8]

2.3.1 RADAR

No sistema RADAR, os melhores resultados foram obtidos a partir da abordagem em *fingerprints*. Esta técnica é caracterizada por duas fases, a fase *offline* e a fase *online* de determinação de posição. Durante a fase *offline,* são efetuadas leituras dos RSS relacionados com os pontos de acesso na vizinhança, posições reais terrestres (x,y,z) e finalmente armazenados esses mesmos valores numa tabela denominada por mapa rádio (*radio map)*. Cada entrada com (local, SSID, RSS) é designada por *fingerprint*. Durante a determinação de posição, *online phase*, são adquiridos, por dispositivos móveis, dados referentes aos pontos de acesso detetados e potências de sinal associadas (RSS). Este *dataset sample* é comparado com as *fingerprints* já presentes no *radio map*, anteriormente elaborado na *offline phase*, e é devolvida a *fringerprint* mais próxima da *sample*, sendo que as coordenadas reais terrestres presentes na *fingerprint* de retorno corresponderão à localização atual. O método como é devolvido esse resultado possui vários algoritmos matemáticos, presentes noutros estudos já efetuados.

2.4 Dynamic Radio Map

Com o tempo apareceram novas implementações onde o *radio map* abordadado em [9] é apenas um mapa estático onde residem *fingerprints* capturadas. Classifica-se como estático pois esses valores são guardados no servidor e apenas voltaram a ser usados para prever localizações e nunca mais atualizados.

Com isto surge um problema comum, abordado anteriormente, na fase *online.* Amostras RSS recebidas variam com o ambiente onde o *radio map* foi construído, isto leva a que pedidos de localização quando o ambiente proporciona condições de propagação diferentes possam ter prognósticos de localizações erráticas. Este problema poderá ser ultrapassado quando se cria um novo mapa rádio, mas esta abordagem irá consumir muito tempo a recolher os dados e posteriormente a tratá-los na construção do mesmo. Consequentemente, um mapa rádio dinâmico poderá resolver os problemas acima referidos. A construção deste novo mapa irá relacionar a relação do RSS medido quando as condições do ambiente estão alteradas com os valores do mapa rádio estático. Usando um modelo que prevê e estima a relação entre os dois ambientes ou intervalos de tempo, consegue-se então criar um mapa dinâmico.

Para se conseguir criar um mapa dinâmico com estas características temos então que criar os designados *calibration points* que são pontos fixos onde são efetuadas leituras em diferentes alturas do dia, mês, estação. Estas leituras realizadas nos pontos de calibração não deixam de ser *fingerprints* mas com o único objetivo de calibração, pois estas irão ditar a relação RSS em t_0 (mapa estático) com a leitura realizado no momento t_i nos variados pontos de calibração.

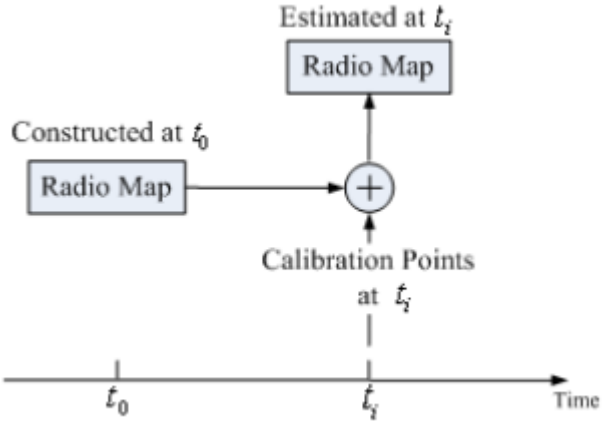


Figura 2.1 - Modelo dinâmico de um *radio map* [10]

Na figura acima é demonstrado o método como é gerado o novo mapa rádio, baseado no mapa rádio estático com a junção dos *calibration points*.

$$r_{jk}(t_i) = f_{jk}(s_{1k}(t_i), s_{2k}(t_i), \dots, s_{Nk}(t_i))$$

Esta equação mostra como podem ser obtidos os valores RSS dos pontos de referência (*fingerprints*) $r_{jk}(1 \leq j \leq M; 1 \leq k \leq P)$ onde M é o número de pontos de referência e P o número de pontos de acesso visíveis. S corresponde ao RSS do ponto de calibração, em que N é determinado pelo número de pontos de calibração disponíveis.

2.5 Algoritmos matemáticos de localização em sistemas RADAR-based

Existem vários algoritmos utilizador para prever a localização do objeto móvel a monitorizar. A primeira, e muito utilizada, função chama-se distância Euclidiana [7], [10], [11] onde a sua expressão calcula a distância "ordinária" entre coordenadas num espaço n. [12]

$$\mathrm{d}(\mathbf{p}, \mathbf{q}) = \mathrm{d}(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

Figura 2.2 - Distância Euclidiana [12]

Outros métodos matemáticos são também utlizados e aplicados nos mapas rádio tais como o KNN (*K nearest neighbor)* onde o seu objetivo é classificação ou regressão. Aplicado no contexto de um *radio map*, o objeto móvel é classificado relativamente às vizinhanças no mapa rádio. Esta classificação calcula K vizinhos e em *majority vote* é determinada qual a sua classe. [13]

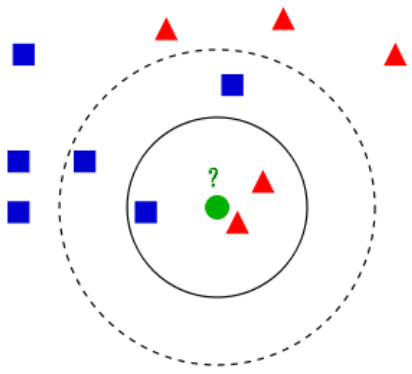


Figura 2.3 - Modelo de classificação k-nn [13]

Já o modelo de regressão kNN é baseado no mesmo princípio mas em vez de classificar um determinado tipo estima um valor inteiro presente nessa variável.

Esse algoritmo é dado pelo demonstrado na equação em baixo.

$$\hat{y} = f(x) = \frac{1}{k}\sum_{j=1}^{k} y_{ij}$$

Este modelo de regressão primeiramente calcula as distâncias Euclidianas dos *training points* x_i vizinhos de x, ordenadas por ordem crescente e selecionados os k vizinhos. Após isto são selecionados os valores dos vizinhos correspondentes $y_{i1} \ldots y_{ik}$ e realizada a média das suas distâncias Euclidianas calculadas anteriormente. [10], [11]

2.5.1 Triangulação (GPS-like Indoor Positioning System)

Em resposta ao modelo RADAR, foi concebido um modelo similar ao GPS [14], que não incorpora a fase *offline* ou treino e não depende de conhecimento anterior do espaço a monitorizar, podendo ser instalado em qualquer organização sobre a sua rede Wi-Fi com mudanças mínimas e custos diminuídos. Este sistema consiste na triangulação de pontos de acesso LAN baseados em modelos híbridos de propagação. Este modelo é suportado por três segmentos: segmento WLAN, segmento de controlo de o segmento do utilizador. O segmento WLAN é caracterizado pelos APs, tal como no GPS os satélites, que dão cobertura ao local e ao mesmo tempo vão executar varrimentos às redes vizinhas, enviando posteriormente essa informação para o segmento de controlo. No segmento de controlo são recebidas as informações recolhidas pelo segmento WLAN, é elaborado o modelo hibrido logarítmico de propagação e finalmente enviado este modelo para o segmento WLAN. Finalmente, o objeto móvel a monitorizar, sendo o segmento do utilizador, executa varrimentos aos APs e recebe, pelo segmento WLAN, os modelos de propagação a fim de prosseguir à trilateração.

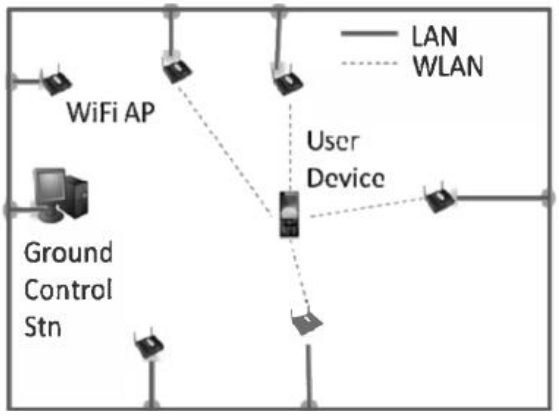


Figura 2.4 - Modelo de triangulação [14]

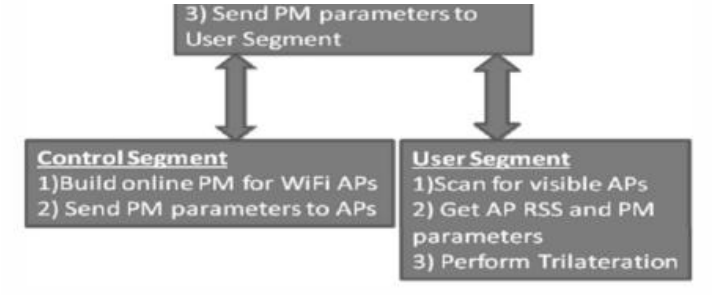


Figura 2.5 - Segmentos do modelo *GPS-like* [14]

Com isto foi demonstrado um sistema sem necessidade de recolha de dados em fases *offline* anteriores e que, ao mesmo tempo, se adapta às constantes mudanças do meio onde se propagam as redes 802.11.

2.6 Pesquisa teórica relativa aos conceitos abordados

2.6.1 Ethernet, IEEE 802.3

A tecnologia Ethernet foi desenvolvida na Xerox PARC entre os anos 1973 e 1974. A empresa Xerox registou uma patente com os nomes Robert Metcalfe, David Boggs, Chuck Thacker, and Butler Lampson como inventores. Este trabalho foi tão notável que o próprio Metcalfe decidiu sair da Xerox e convencer a Digital Equipment Corporation (DEC), Intel e a Xerox para trabalharem em conjunto de modo a promover a Ethernet como um *standard*.

O chamado "DIX" *standard* resultado de "Digital/Intel/Xerox" é especificado com velocidades até 10 Mbit/s Ethernet, endereços de destino e origem compostos por 48-bit e um campo global de 16-bit denominado por Ethertype-type. A “Version 2” foi publicada em Novembro de 1982 e definiu o que ficou conhecido pela Ethernet II, resultado de uma publicação do IEEE 802.3 em Junho de 1983.

Antes da existência da Ethernet existiam alguns problemas no que toca a redes distribuídas. Alguns exemplos foram listados abaixo.

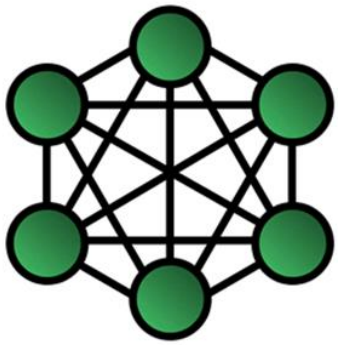
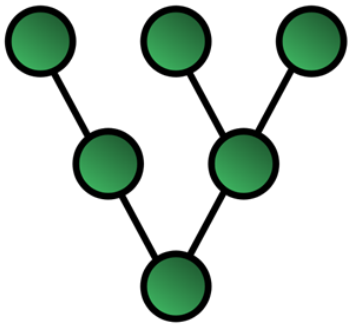
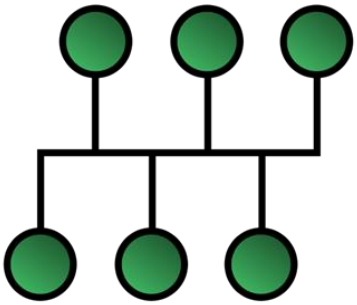
		
Exagerado meio físico	Partilhar um único meio físico sem moderá-lo. Aumentaria a utilização neste meio resultando num aumento de vulnerabilidades aos *single points of failure*	A topologia em barramento falha no caso de o utilizar querer adicionar muitas máquinas à rede

Tabela 1 - Topologias de redes distribuídas

Com isto podemos concluir que moderar o acesso à estrutura física poderá ser uma vantagem, por exemplo performance.

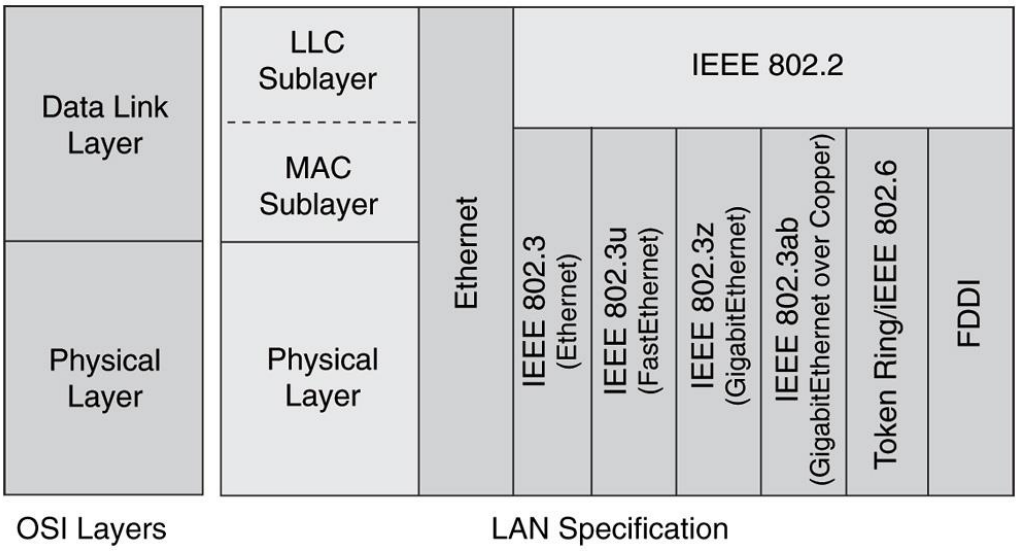


Figura 2.6 - A localização do *standard* Ethernet no modelo OSI [15]

2.6.1.1 Camada física (Physical Layer)

O termo Ethernet refere-se à família dos produtos *local-area network* (LAN) cobertos pelo *standard* IEEE 802.3 definindo nele o que é mais conhecido pelo protocolo CSMA/CD. Existem três taxas de transferências de dados atualmente definidas para operar sobre fibra ótica ou cabos de par entrançado:

- 10 Mbps-10Base-T Ethernet
- 100 Mbps-Fast Ethernet
- 1000 Mbps-Gigabit Ethernet

A Ethernet sobreviveu como a principal tecnologia abrangendo as redes LAN porque a sua estrutura possui as seguintes características:

- Fácil de entender, implementar, gerir e efetuar manutenção
- Permite uma implementação *low-cost*
- Fornece uma extensiva flexibilidade topológica na instalação deste tipo de redes
- Garante conexões bem-sucedidas entre máquinas sendo que estas operam sobre um *standard* e não sobre produtos de fabricantes específicos. Isto significa uma especificação genérica sem dependências fabris. [16]

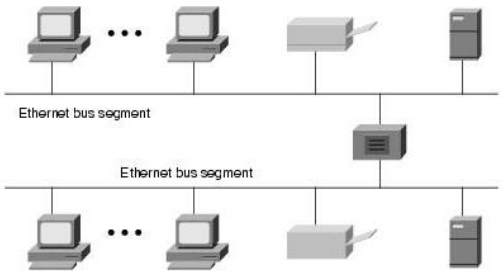
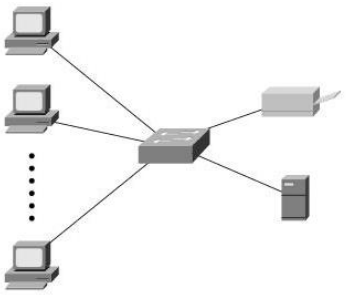
		
Ponto-a-ponto	Barramento Coaxial	Estrela

Tabela 2 – Nível físico - Topologias de rede [16]

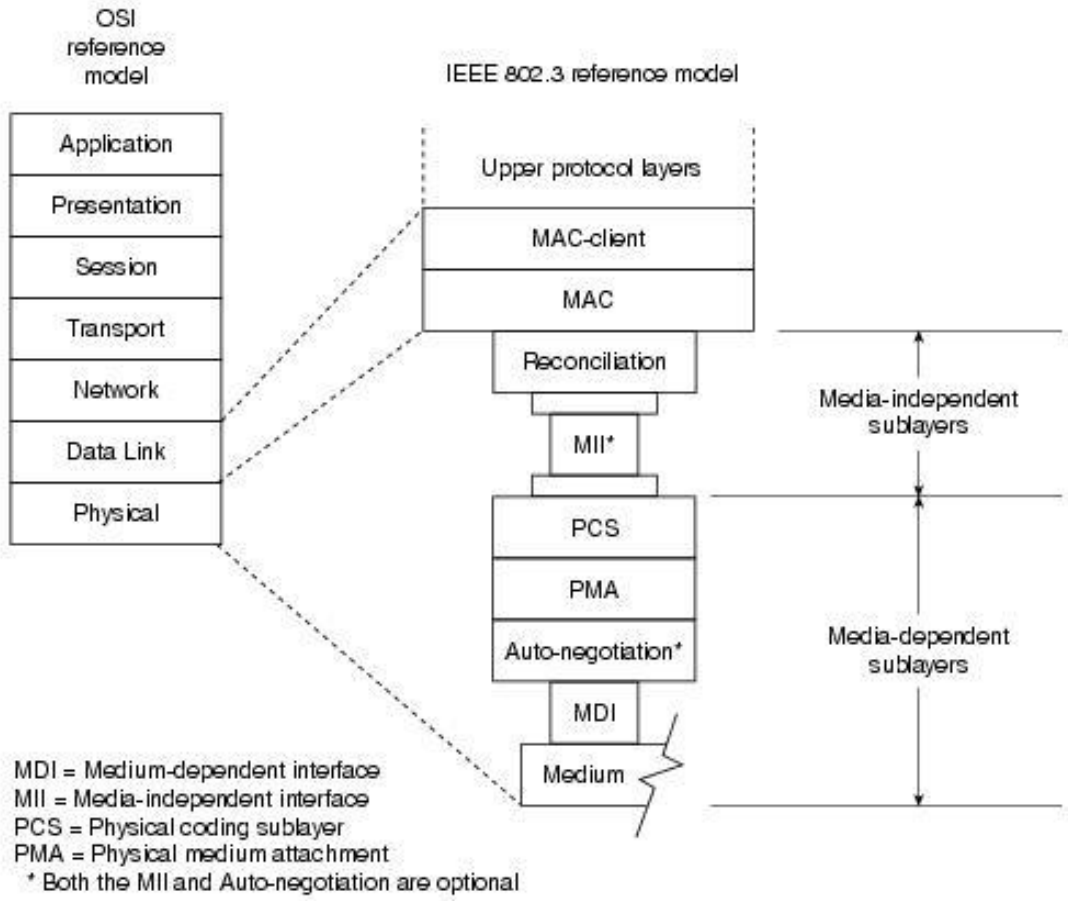


Figura 2.7 - Camada física - Relação do *standard* com o modelo de referência OSI [16]

2.6.1.2 Camada de ligação (Data link Layer)

A camada de ligação fornece os meios e procedimentos funcionais para transferir dados entre entidades de presentes numa rede e pode também fornecer meios para detetar e possivelmente corrigir erros que podem ocorrer na cama física. [17]

A cama de ligação fornece:

- Encapsulamento dos pacotes de dados da cama de rede em chamadas *frames*
- Sincronização entre *frames*
- A subcamada Logical Link Control (LLC) suporta:
    - Controlo de erros (pedido automático de retransmissão, ARQ)
    - Controlo de fluxo

Subcamada de controlo de acesso ao meio (MAC):

- Protocolo de múltiplo acesso para o controlo de acesso ao canal, por exemplo o protocolo CSMA/CD responsável por detetar colisões e retransmitir no *standard* nas topologias de rede Ethernet em barramento e hub

- Endereçamento físico (endereçamento MAC)
- LAN *switching* (*packet switching*) incluindo filtragem MAC

Subcamada "Media access control" (MAC)

A subcamada MAC fornece mecanismos de endereçamento e controlo de acesso ao meio fazendo com que seja possível que numa rede com vários terminais e nós possam comunicar entre si partilhando o mesmo meio físico. [18]

Funções realizadas pelo MAC MAC:

- Receber e enviar *frames* normais
- Retransmissão *half-duplex* e funções de *backoff*
- Incluir e verificar o campo FCS (frame check sequence)
- Aplicação de um *interframe gap*
- Descartar pacotes com erros
- Anexar ou remover o preamble, SFD (Start Frame Delimiter), e *padding*
- Compatibilidade *half-duplex*: anexar ou remover endereços MAC

O que acontece quando as estações A e B enviam dados para a outra? Como detetar as colisões?

Figura 2.8 - Colisão entre duas máquinas

Se o meio físico estiver ocupado, o transmissor espera pelo meio ficar desocupado e logo depois inicia a transmissão.

Enquanto uma estação está a transmitir dados, a mesma lê as diferenças de potencial nessa linha tentando verificar se existem interferências, resultando em potenciais diferentes, ou seja, colisões.

O *standard* IEEE 802.3 especifica o valor máximo do tempo de transmissão numa rede 10BaseT (no pior cenário - 10Mbps):

- Distância máxima entre máquinas deverá ser no máximo 2500 metros.
- Á velocidade de 10Mbps um único bit demora 0.1µs a ser transmitido. Posto isto, as frames Ethernet devem conter no mínimo um comprimento de 64 bytes
- 14 bytes cabeçalho (6*2 bytes para endereços de destino e origem + 2 bytes EtherType)
- 46 bytes dados (*payload*)
- 4 bytes CRC

$$64 * 8 * 0.1\mu s = 51.2\mu s$$

Os tempos de atraso durante retransmissões são selecionados a partir de um backoff binário exponencial.

$$delay = 51.2\mu * random(0 \dots 2^{n-1} - 1)$$

- A variável "n" significa o número de tentativas depois de uma deteção de colisão
- O atraso é expresso em segundos
- O método random é caracterizado como um algoritmo não determinístico, assegurando que não existem colisões em loop infinito.

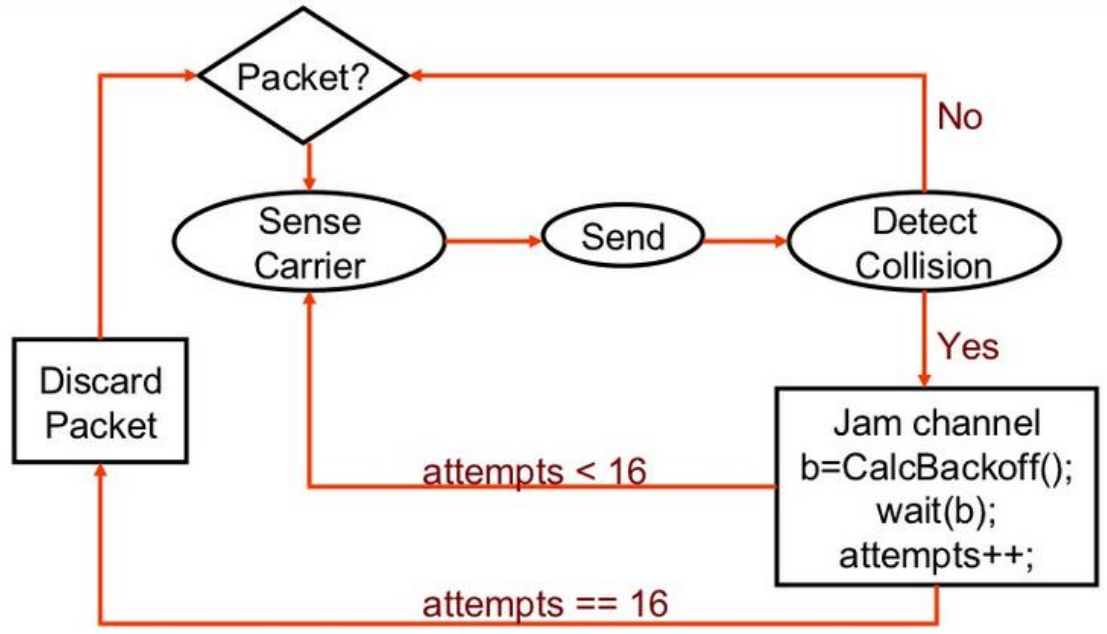


Figura 2.9 - Algoritmo de *backoff*

Subcamada" Logical link Control" (LLC)

A camada superior protocolar de comunicação de dados LLC, presente na cama de ligação do modelo OSI, é conhecida também pela camada 2. Esta subcamada fornece mecanismos de multiplexagem que fazem com que seja possível que vários tipos de protocolos numa rede multiponto coexistam sobre o mesmo meio físico. [19]

A subcamada LLC é primariamente responsável por:

- Multiplexagem de protocolos a ser transmitidos sobre a camada MAC na transmissão e descodificação dos mesmos na receção.
- Fornecer controlos de fluxo e erros entre nós

2.6.1.3 Ethernet *frame*

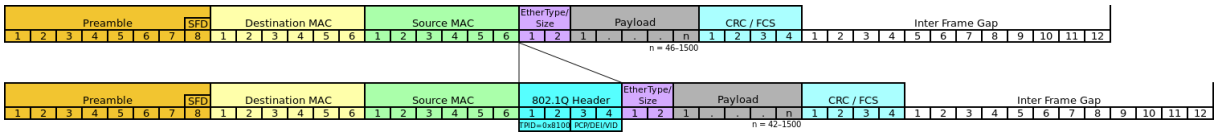
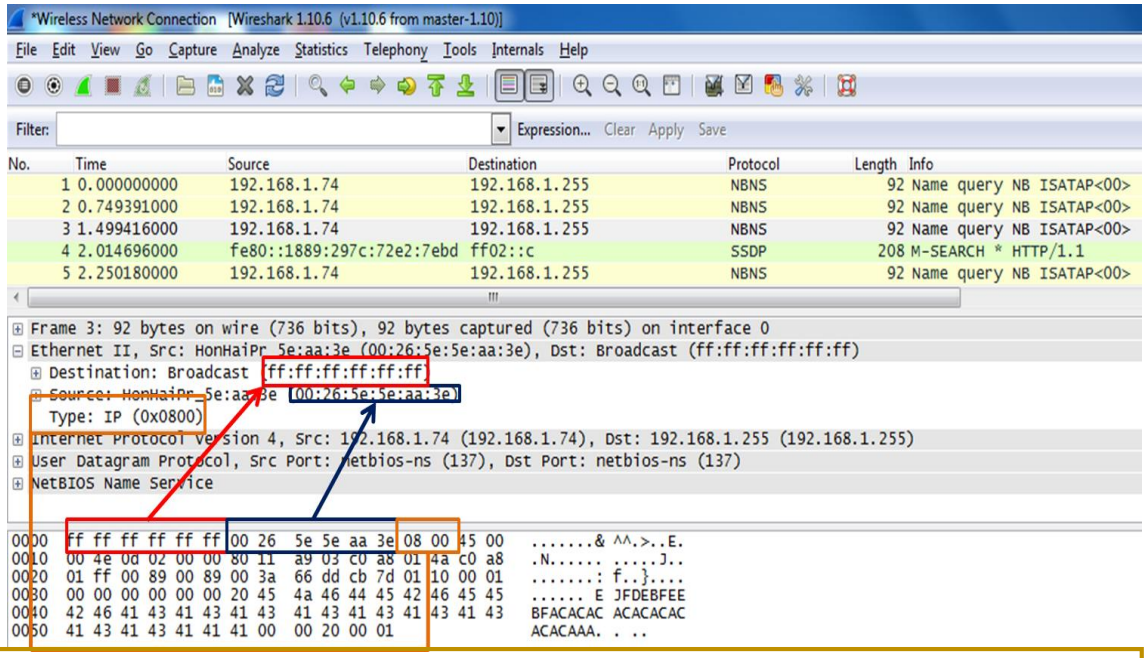


Figura 2.10 - Cabeçalho Ethernet e seus campos subdivididos

- Preamble (8 bytes) – 1010…1011. Usado para sincronizar o recetor e as taxas de envio do transmissor
- CRC (4 bytes) – Confirmado pelo recetor, caso algum erro seja detetado a *frame* é ignorada
- Destination/Source MAC (6 bytes) – Endereços relativos à origem e destino do pacote
- EtherType (2 bytes) – Campo usado para indicar qual o protocolo que está encapsulado no *payload* da *frame* Ethernet
- Payload – Dados da *frame*. O *payload* mínimo é de 42 bytes (se presente o cabeçalho 802.1Q) ou 46 bytes. *Payload* máximo é composto por 1500 octetos.
- Frame check sequence (FCS) é um campo de verificação de redundância cíclica composta por 4 bytes que permite a deteção de dados corrompidos pela *frame* completa.

2.6.1.4 Wireshark – Exemplo prático



O valor presente no campo EtherType indica qual o protocolo encapsulado no *payload* desta *frame* Etherned II, como anteriormente explicado. Neste exemplo o valor 0x0800 listado no IEEE *standard* corresponde ao protocolo Internet Protocol version 4 (IPv4) que por sua vez corresponde exatamente ao protocolo da *frame* delimitada pelo wireshark.

2.6.2 Wi-Fi, IEEE 802.11

Conhecida por *wireless* LAN, seu acrónimo WLAN, ou WiFi/Wi-Fi (Wireless Fidelity) é uma tecnologia que permite a transmissão de dados a partir de um sistema sem fios. Esta invenção foi desenhada para fornecer acesso a uma rede de computadores que comuniquem entre si usando ondas radio trabalhando em bandas de frequência a 2.4,3.6,5 e 60 GHz) substituindo assim uma infraestrutura por cabo.

Para que se consiga acompanhar o avanço da tecnologia com a amplificação das taxas de transmissão de dados, largura de banda, *throuput*, redução dos custos de produção como também a adição da tecnologia *multiple-input multiple-output antennas* (MIMO), várias especificações foram criadas ao longo destes anos. [20]

Estas são as especificações mais comuns que existem até aos dias de hoje.

Designação do *standard*	Data de homologação	Banda	Taxa de transmissão de dados	Modulação / acesso
802.11	1997	2.4Ghz	1-2 Mbit/s	FHSS, DSSS, CCK
802.11a	1999	5 Ghz	54 Mbit/s	OFDM, BPSK, QPSK, QAM
802.11b	1999	2.4 Ghz	11 e 5.5Mbit/s	DSSS, CCK
802.11g	2002	2.4 Ghz	54 Mbit/s	OFDM, BPSK, QPSK, QAM
802.11n	2007	2.5 Ghz e 5Ghz	100 até 320 Mbit/s	OFDM, MIMO

Tabela 3 - Especificações 802.11

As redes 802.11 WLAN são compostas por:

Serviços fornecidos pela rede:

- Autenticação e de autenticação – série de testes para confirmar a autenticidade do dispositivo;
- Privacidade – algoritmo de encriptação usado para proteger estes pontos de acesso sem-fios por parte de acessos externos (WEP, WPA, WPA2);
- Entrega *MAC Service Data Unit* (MSDU) – garante que esta unidade é transferida pelo serviço MAC do ponto de acesso

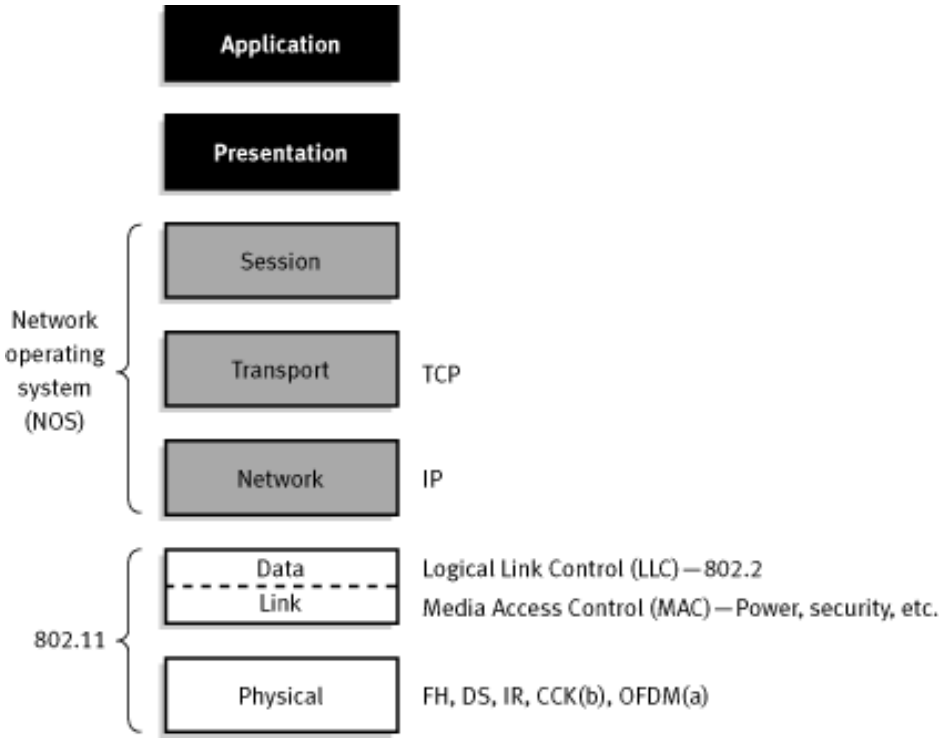


Figura 2.11 - 802.11 no modelo OSI [33]

2.6.2.1 Camada física (*Physical Layer*)

Com o espectro definido a 2.4 Ghz e 5 Ghz, o *standard* Wi-Fi é designado por ser bastante robusto a interferências, mantém a velocidade das altas taxas de transmissão de dados e pelas especificidades de segurança.

Cumprindo estes, algumas técnicas de modelação foram inicialmente adotadas:

- Infravermelhos (eliminada pelas restrições de alcance);
- Frequency Hopping Spread Spectrum Technique (FHSS);
- Direct Sequence Spread Spectrum Technique (DSSS).

Mais tarde estas tecnologias mostraram-se incapazes de ultrapassar velocidades de transmissão de 2Mbps, outras técnicas foram implementadas:

- Complementary Code Keying (CCK) técnica de modelação de dados (velocidades acima dos 11Mbps);
- Orthogonal frequency-division multiplexing (OFDM) and Multiple-Input Multiple-Output OFDM (MIMO-OFDM) (acima dos 54Mbps).

2.6.2.2 Camada de ligação (*Data link Layer*)

Tal como no *standard* IEEE 802.3 em 2.6.1.2 , a camada de ligação fornece os meios e procedimentos funcionais para transferir dados entre entidades de presentes numa rede e pode

também fornecer meios para detetar e possivelmente corrigir erros que podem ocorrer na cama física.

Subcamadas MAC e LLC

O *Medium Access Layer* pode ser gerido dois métodos:

- **Distributed Coordinated Function (DCF):**
  - Simétricos, todas as estações (incluindo os pontos de acesso) comportam-se do mesmo modo;
  - Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA);
  - Estações disputam pelo acesso ao meio.

- **Point Coordinated Function (PCF):**
  - Construído no DCF;
  - Permite períodos livres de disputa intervalados com períodos de disputa;
  - Uma estação (tipicamente um AP) executa verificação com saldos, verificando quem transmite;
  - Permite operações mais eficientes quando existem cenários de maior exigência de processamento.

Relativamente à subcamada LLC, esta funciona exatamente da mesma maneira que no tópico 0**Subcamada" Logical link Control" (LLC)**.

2.6.2.3 802.11 frame

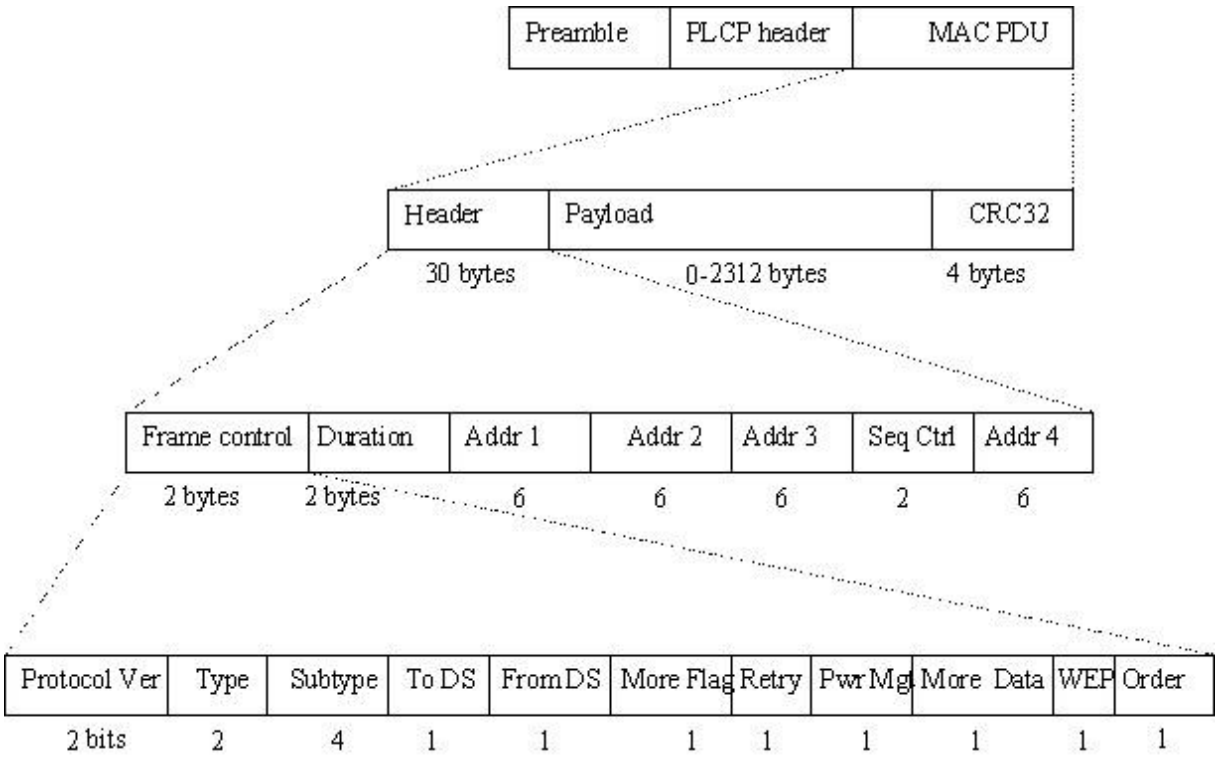


Figura 2.12 - Pacote IEEE 802.11 e subdividido em cabeçalhos [34]

2.6.3 Camada de rede (Network Layer) – Internet Protocol (IP)

O *Internet Protocol* (IP) é um protocolo inserido na camada que rede que consiste em endereçamento e controlo de informação para que os pacotes possam ser roteados dentro de uma rede. Este *layer* de rede é considerada uma das camadas mais notáveis presentes no modelo OSI visto que seleciona e gere o melhor caminho lógico para transferir dados entre nós.

Em contraste com o *Data link Layer*, que lida somente com máquinas locais, a *Network Layer* é a camada mais baixa, presente no modelo OSI, que se preocupa com o envio de dados entre máquinas mesmo que estas sejam remotas, não-locais. O estabelecimento da conexão lógia, encaminhamento de dados, roteamento e entrega de relatórios de erros são as responsabilidades primárias desta cama.

Nesta camada está inserido *hardware* como *routers, bridges, firewalls* e *switches.* Para suportar o roteamento, a *Network Layer* não só administra endereços lógicos, tais como os endereços IP, como também gere o mapeamento entre estes endereços lógicos e físicos.

Entre outros, estas são as operações normais efetuadas por esta camada:

- Endereçamento lógico – Cada dispositivo numa rede tem um endereço lógico, que é independente do *hardware* existente nessa máquina, e este deve ser único dentro de toda a rede.
- Encapsulamento de datagramas – Encapsulamento de mensagens recebidas das camadas superiores, colocando-as em datagramas, também denominados por pacotes, com um cabeçalho adicional, conhecido por *header*.
- Roteamento – Serviço que direciona pacotes para a máquina de destino. As máquinas de origem e destino nem sempre estão conectadas à mesma rede, fazendo com que o pacote tenha que ser guiado pela rede, chegando assim ao destino final.
- Desencapsulamento – Examinação realizada pelo endereço de destino que verifica se o endereço que o pacote contém corresponde ao endereço da máquina, se correto, o pacote é desencapsulado pela camada de rede.

O protocolo IP permite o utilizador em endereçar um pacote e envia-lo para a rede, mas não existe nenhum mecanismo de ligação direta entre ele e o destinatário. Posto isto, este protocolo, imensamente usados na internet, opera juntamente com o *Transport Control Protocol* (TCP), referenciado como TCP/IP, podendo ser utilizado em cima de diferentes *data link interfaces* tais como Ethernet ou Wi-Fi.

2.6.3.1 Internet Protocol Version 4 (IPv4)

Os dados num pacote IP são encapsulados e organizados em cada um destes parâmetros.

Um *header* é composto por:

- Origem (*source*)
- Destino (*destination)*
- Outras informações relativas ao *payload* do pacote

Mensagem de dados (*payload)*

A versão mais usada até aos dias de hoje foi, sem dúvida, a versão IPv4, contudo, o seu sucessor, *Internet Protocol Version 6* (IPv6) começa agora a ser suportado.

Características básicas do protocolo IPv4:

Sem necessidade de executar conexão – Não existe conexão lógica nem física antes do envio de datagramas. Os pacotes IP são enviados sem que a máquina de origem avise o

destinatário, portanto este protocolo não necessita de troca ou acordo inicial entre *hosts* para se estabelecer uma conexão *end-to-end* antes que os pacotes sejam reencaminhados.

Melhor esforço (não confiável) – Não existe *overhead* associado à transmissão, não garantindo assim a chegada do datagrama. Não confiável surge porque não havendo garantias de chegada do datagrama, isto faz com que este protocolo não consiga gerir ou reenviar pacotes não recebidos ou corrompidos. Outros protocolos, noutros níveis, conseguem gerir estas situações.

Independente do meio – Opera sem dependência do meio em que os dados são transmitidos, significando assim que o protocolo IP pode ser enviado eletricamente por cabo, sinal ótico por fibra ou sem fios através de sinais rádio.

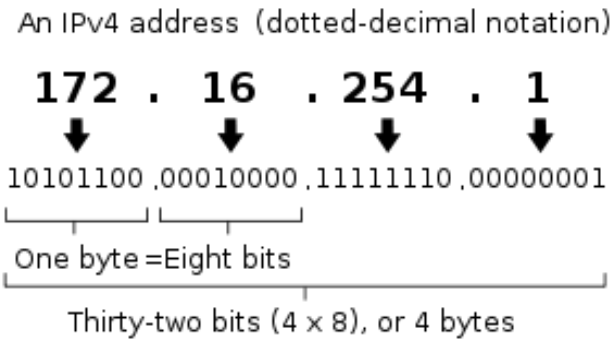


Figura 2.13 – Endereçamento IPv4 [21]

Na figura acima é apresentado o modelo de representação de um endereço IPv4. Esta representação está agrupada em quatro octetos, perfazendo dois bytes ou 32-bit. A representação por pontos representa o número decimal do octeto em que este se insere.

Classe	Gama de Endereços	Nº de Endereços por Rede
A	1.0.0.0 até 127.0.0.0	16 777 216
B	128.0.0.0 até 191.255.0.0	65 536
C	192.0.0.0 até 223.255.255.0	256
D	224.0.0.0 até 239.255.255.255	Multicast
E	240.0.0.0 até 255.255.255.254	Uso futuro; reservada a testes pela IETF

Os formatos dos endereços IP são classificados em diversas classes, podendo ser divididos em redes mais pequenas, conhecidas por *subnetworks* ou *subnets*. Estas são criadas com ajuda das *IP Subnet Masks*. Estas são criadas para aumentar a flexibilidade do sistema, uso

mais eficiente da rede e a capacidade de redirecionar trafico *broadcast* apenas para uma determinada sub-rede.

2.6.3.2 Internet Protocol Version 6 (IPv6)

Para colmatar algumas limitações do protocolo IPv4 relativas ao número de combinações possíveis que o comprimento dos seus endereços poderiam administrar, foi assim criado o protocolo IPv6. Esta invenção usa 128-bits (16-bytes) de endereços que permitem a existência de $2^{128} = \sim 3.4 * 10^{38}$ endereços IP diferentes [21], contrariamente aos 4-bytes usados no protoloco IPv4.

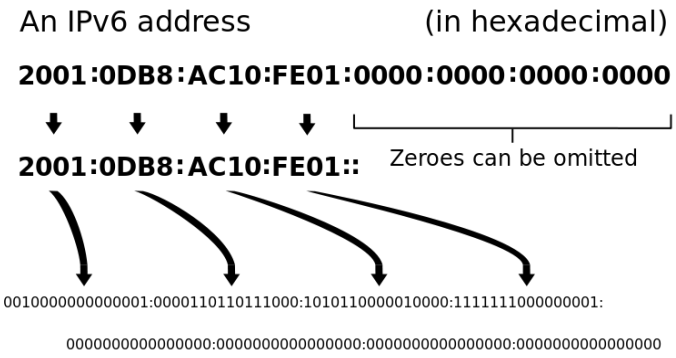


Figura 2.14 - Endereçamento IPv6 [21]

2.6.3.3 Cabeçalhos IP

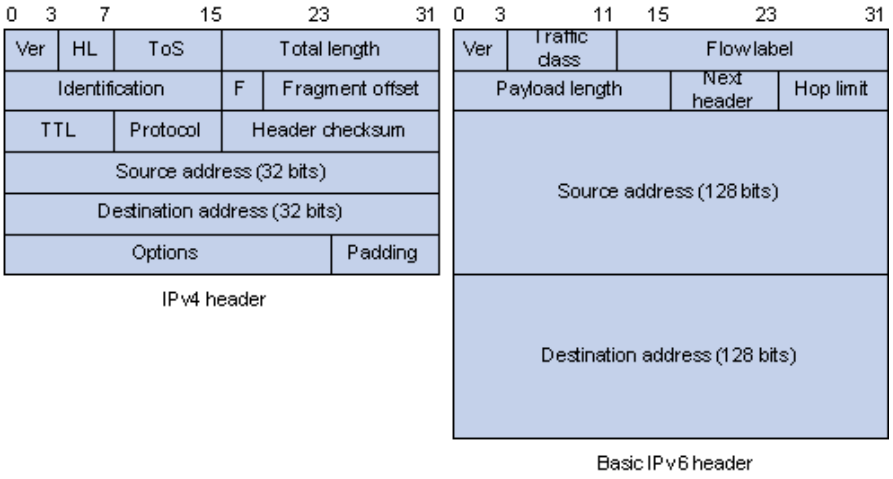


Figura 2.15 - Comparação *headers* IPv4 vs IPv6 [35]

2.6.4 Camada de transporte – (TCP, UDP)

A cama de transporte fornece serviços de comunicação *end-to-end* ou *host-to-host* para aplicações dentro de uma arquitetura em camadas de componentes e protocolos de rede. [22]

A camada de transporte fornece:

- Serviços orientados à conexão
- Confiabilidade
- Controlo de fluxo
- Multiplexagem
- O TCP (*Transmission Control Protocol*) e o UDP (*User Datagram Protocol*) estão localizados na camada de transporte do modelo OSI
- TCP permite que seja estabelecida uma conexão e troca de dados entre dois (orientado à conexão)
- UDP é um protocolo sem conexão que, como o TCP, trabalha por cima de uma rede IP (orientado à transação)

2.6.4.1 TCP

Vantagens:

- O TCP garante que a data chega ao destino final e ordenada, fazendo com que não sejam enviados pacotes duplicados
- Controlo de fluxo e congestionamento
- Os programadores só se preocupam com os pacotes no chamado *user space* porque a reassemblagem, *acknowledgements*, controlo de fluxo e receção são realizados no kernel.
- *Throughput* relativamente bom

Desvantagens:

- O TCP não conclui uma transmissão sem que todos os pacotes não tenham sido confirmados por *acknowledgements*
- TCP não pode ser usado para técnicas de *broadcast* ou transmissões *multicast* [23]

2.6.4.2 UDP

Vantagens:

- Possível usar as técnicas de *multicast* e *broadcast*

- A conexão não é baseada num modelo de comunicação, significando que a latência e os tempos de operação do sistema são menores, fazendo deste um protocolo mais rápido.

Desvantangens:

- Não é garantida a receção do pacote, receção duplicada ou se a chegada dos pacotes chega com a ordem trocada.
- Não existe controlo de fluxo e congestionamento [24]

Num cenário prático o TCP é mais usado em serviços HTTP, HTTPs, FTP, Telnet, entre outros, o UDP é usado em DNS, DHCP, VOIP, jogos online, *streaming*, etc. Concluindo o que foi acabado de apresentar relativamente às vantagens e desvantagens destes dois protocolos, o TCP é geralmente uma melhor opção mesmo estado associado a *overhead* e consequentes atrasos. Já o UDP é um boa opção para multimédia como o VoIP fornecendo pouco *jitter* a nível de comunicação. [25]

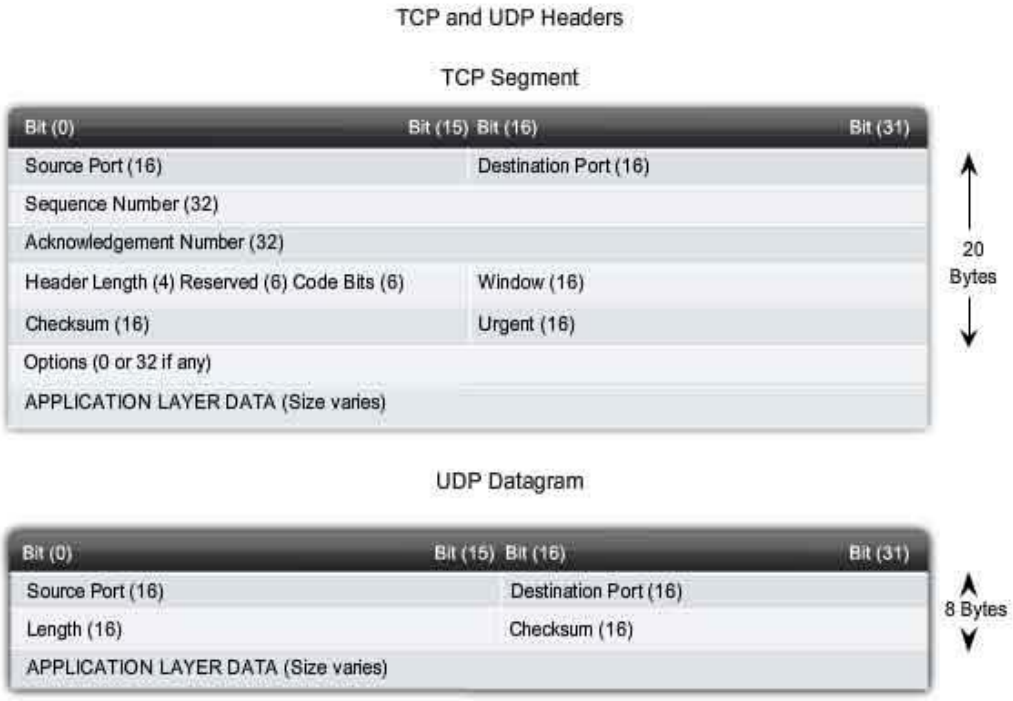


Figura 2.16 - Diferenças entre pacotes TCP e UDP [36]

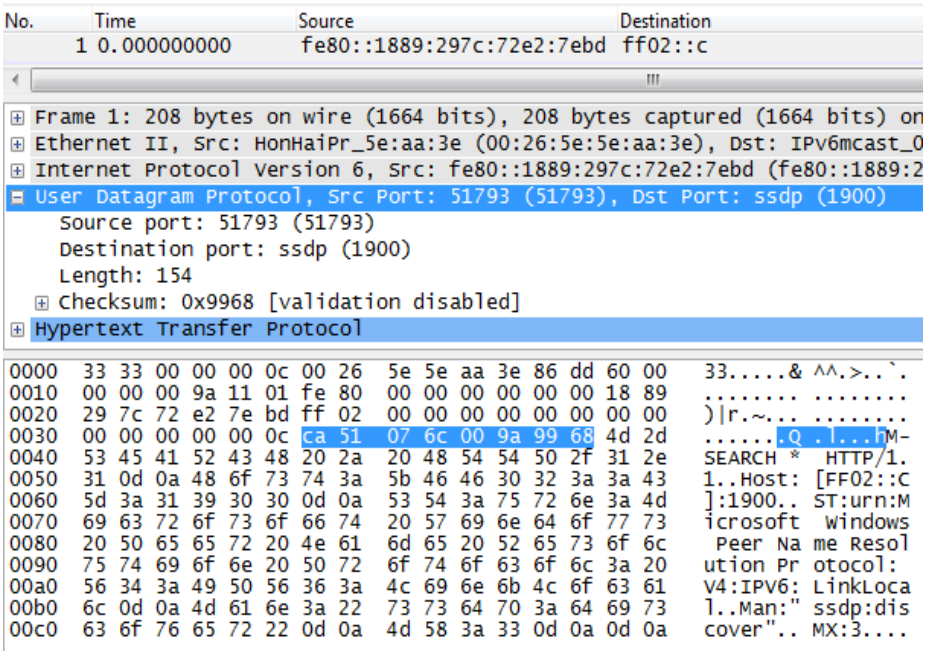


Figura 2.17 - Exemplo prático - Receção pacote UDP e seus parâmetros

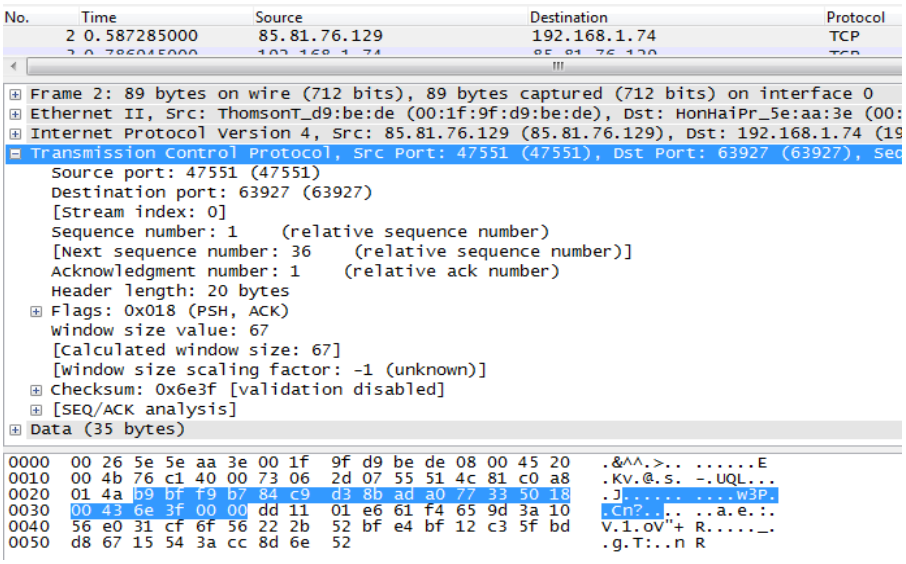


Figura 2.18 - Exemplo prático - Receção pacote TCP e seus parâmetros

Como podemos reparar, em ambas as figuras acima, os cabeçalhos dos segmentos UDP e TCP são bastante distintos em termos de tamanho, comprovando-se também o que é demostrado na Figura 2.16.

2.6.5 Mbed

A plataforma mbed é uma *framework* utilizada para desenvolver aplicações em sistemas embebidos baseadas em microcontroladores ARM. Esta ferramenta foi desenhada para fornecer soluções altamente produtivas para uma rápida prototipagem e desenvolvimento de produto. Este IDE está a ser desenvolvido pela ARM, onde os seus parceiros contribuem globalmente para a comunidade mbed Developer.

A ferramenta Mbed consiste num SDK (Software Development Kit), um HDK (Hardware Development Kit) e ferramentas *online* de desenvolvimento de *software* que integram um conjunto de bibliotecas *open source* com soluções de *design* de *hardware*. [26]

Mbed SDK:

C/C++ SDK para microcontroladores ARM

- APis de alto nível e ambiente *standard*
- Abstração do *hardware*
- Controlo de baixo nível, se necessário
- mbedRTOS e bibliotecas de redes

Baseado nos *standards* das tecnologias presentes na indústria

- ANSI/ISO C/C++
- CMSIS Compliant
- Compatível com grande parte das ferramentas profissionais para MCU
    - Utilizando um compilador *online mbed*

Open Source

- Lançado sobre a licença Apache 2.0
- Adequado para uso comercial e não comercial
- Gerido, supervisionado, testado e sustentado pela ARM

Mbed HDK:

HDK for ARM Microcontroller Boards

- MCU *sub-system* e arquitectura de *debug interface*
- Inclui *on-board* USB interface com três *endpoints*:
    - Programador FLASH Drag-and-drop
    - Porta série virtual VSP (*Virtual Serial Port)*
    - *Debug*

- Permite a ligação por conexões standard em placas *low*-cost, *kits* de desenvolvimento e outros modulos

Baseado nos *standards* das tecnologias presentes na indústria

- USB
- CMSIS-DAP Debug Interface Protocol
- Compatível com Windows, Mac, Linux

Livre para uso comercial e não comercial

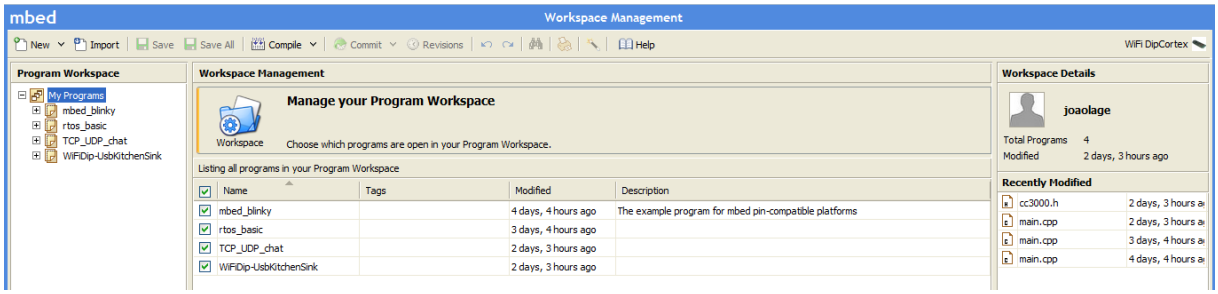
- Gerido, supervisionado, testado e sustentado pela ARM

2.6.5.1 Tour mbed

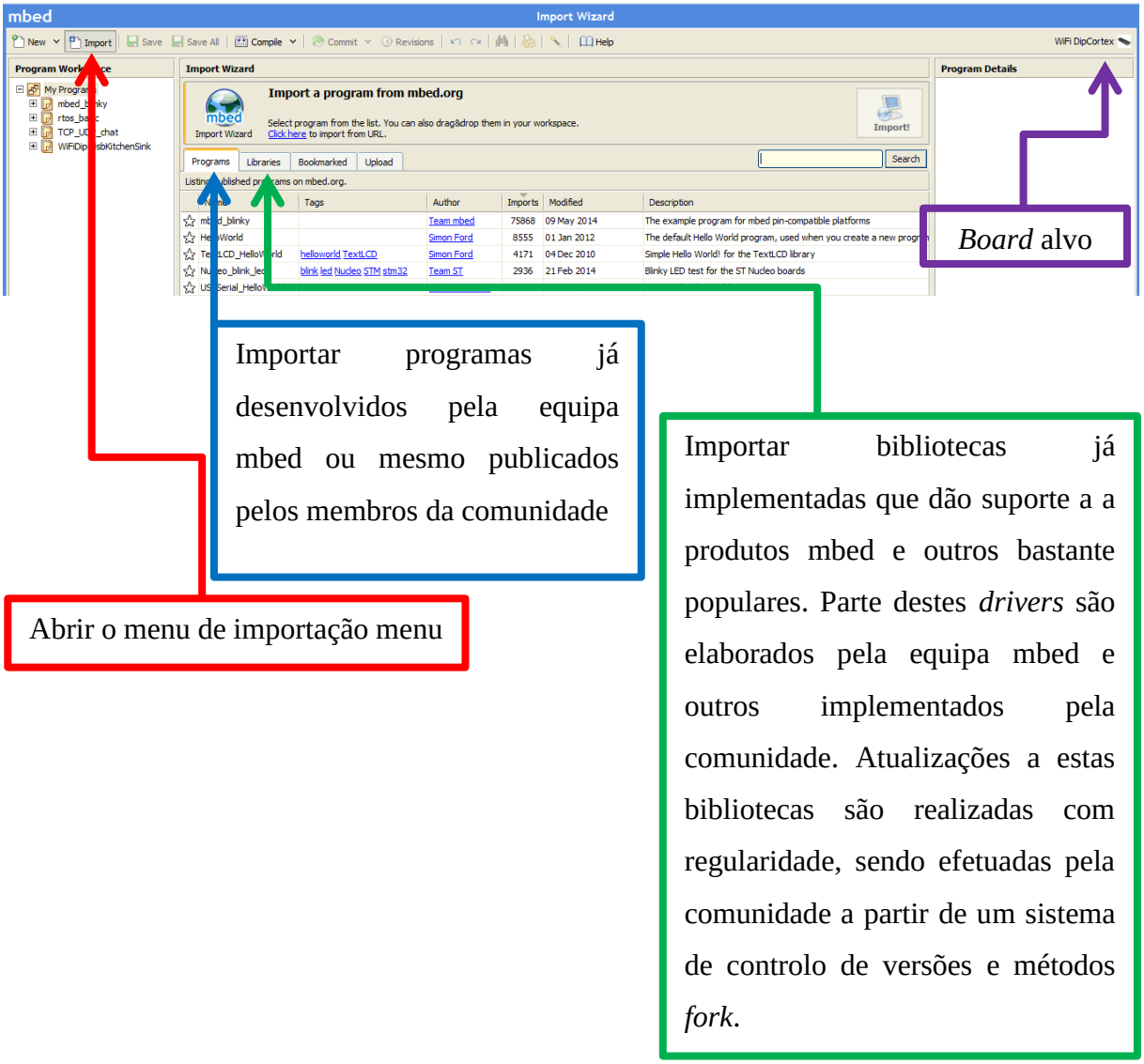
1. Criar uma conta e fazer *login* em www.mbed.org

2. Entrar no compilador *online* da mbed

Aspeto genérico do *workspace*

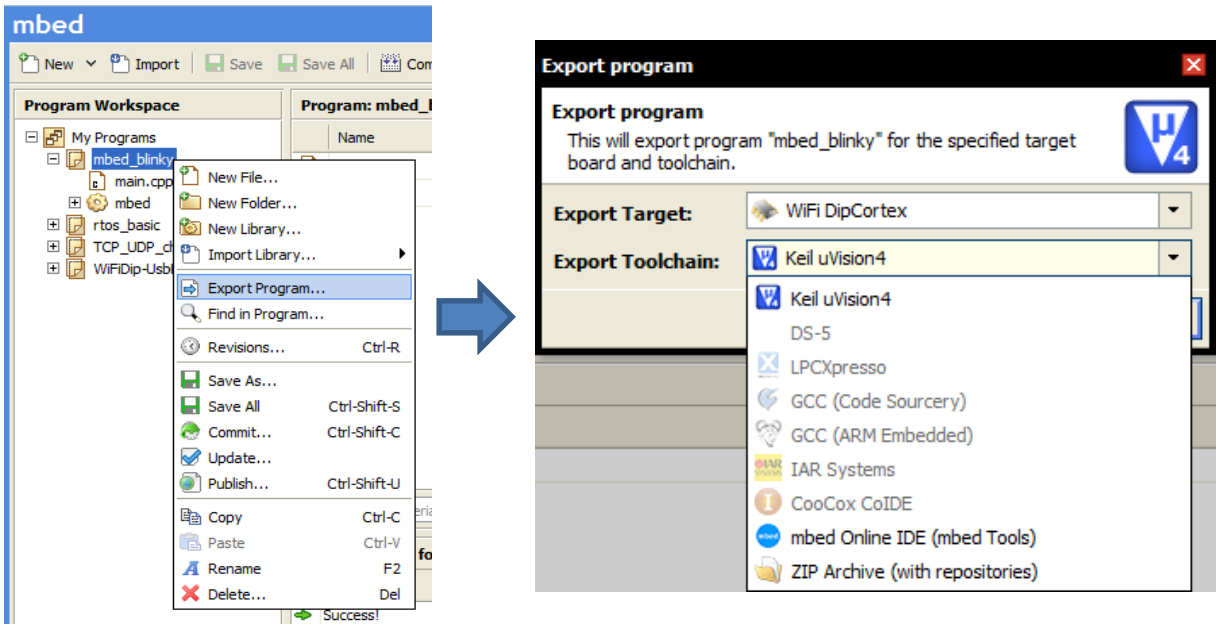


Importação

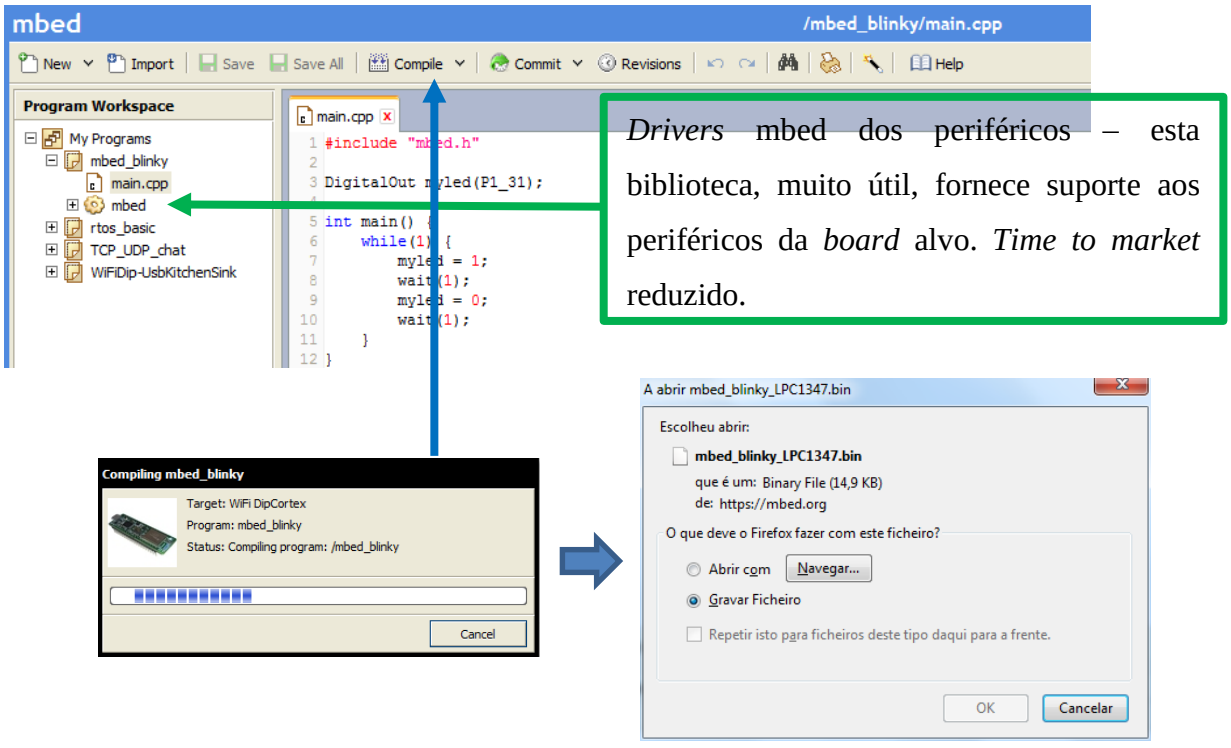


Exportação

Relativamente à exportação, esta função permite ao utilizador conseguir exportar o seu projeto importado ou criado pela plataforma *web* e depois exporta-lo numa pasta comprimida ou até mesmo em formato projeto para o IDE escolhido. A imagem seguinte mostra um projeto importado através do *wizzard* no *browser compiler* e as opções de exportação suportadas. Neste caso temos apenas a opção em pasta comprimida, projeto mbed ou projeto Keil µVision por questões de compatibilidade.



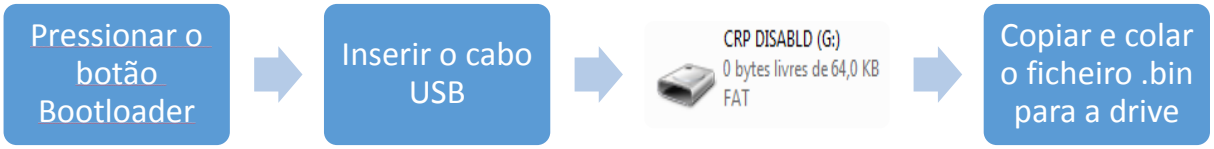
Edição e compilação



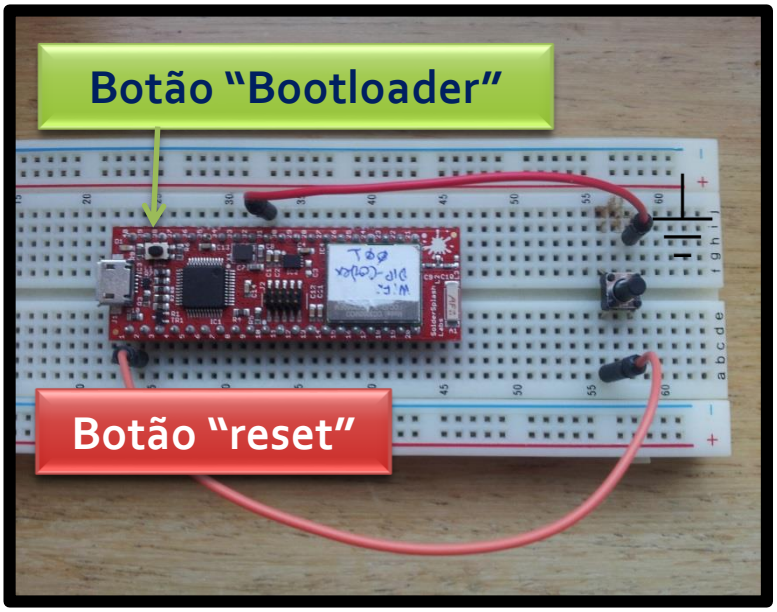
Drivers mbed dos periféricos – esta biblioteca, muito útil, fornece suporte aos periféricos da *board* alvo. *Time to market* reduzido.

Programar um produto mbed

Primeiramente temos que transferir o ficheiro binário (.bin) para a máquina onde irá estar ligada o produto mbed alvo. Depois teremos que, com a sequência demonstrada em baixo, proceder à transferência do ficheiro binário para a *flash* do microcontrolador.



Após este passo ter sido realizado é necessário proceder a um *reset* da placa e então a aplicação é executada. Para executar este passo basta conectar diretamente um botão que realize o contacto entre o *pin reset* e a massa quando pressionado isto porque o *pin reset* funciona em *active-low*.



Informação extra

Como já mencionado, a comunidade mbed oferece uma vasta gama de bibliotecas poderosas que poupam muito tempo ao programador e reduzem drasticamente o tempo necessário para a implementação de um projeto. Algumas dessas bibliotecas notáveis são:

• mbed-rtos (Real Time Operating System produzido pela mbed):
https://mbed.org/users/mbed_official/code/mbed-rtos/

• Ethernet Interface (Implementação da camada IP):
https://mbed.org/users/mbed_official/code/EthernetInterface/

• CC3000 hostdriver with socket interface included:
https://mbed.org/users/Kojto/code/cc3000_hostdriver_mbedsocket/ (*Driver* que integra a comunicação com o módulo CC3000, suas configurações e agrega o uso dos sockets, útil especialmente com a WiFi DipCortex).

2.7 Conclusão do Estado da Arte

Após terem sido demonstradas várias aplicações e investigações realizadas no âmbito da localização em ambientes NLoS podemos concluir que todas estas soluções visam cada vez mais a preocupação em aumentar a precisão da localização sobre a tecnologia apoiada no *standard* IEEE 802.11. De todo o modo concluiu-se, na perspetiva do autor em [14], que, até os dias de hoje, não existe nenhum sistema de localização *indoor* que cumpra as três características base da tecnologia GPS, sendo estas precisão, universalidade e custo reduzido.

Concluindo este tópico e localizando o trabalho a montante, este enquadra um sistema baseado no sistema RADAR onde são adquiridas e enviadas, no modo *offline*, vários *datasets* de *fingerprints* para que se consiga elaborar um mapa rádio presente num servidor alojado na rede. O estudo e implementação da aquisição e envio das *fingerprints* foram realizadas apenas neste relatório e a receção e monitorização desses mesmos dados foram elaboradas em paralelo numa diferenciada dissertação de mestrado.

Capítulo 3 Análise do sistema

Este capítulo irá incluir uma reflexão baseada numa breve análise do sistema tendo como objetivo final descrever a melhor metodologia a usar para as fases posteriores. Posto isto, deve ser realizada uma análise do sistema onde esta deve conter todos os requisitos propostos para que o resultado final seja o esperado e deste modo haverá uma evolução contínua no projeto, sem paragens e redefinição de requisitos. Finalizar-se-á este capítulo com a junção de todos os temas abordados e uma imagem generalizada do sistema.

3.1 Requisitos do sistema

Neste subcapítulo iremos abordar não só os requisitos diretamente impostos como também aqueles que, não tendo sido pedidos e partiram de decisões próprias, foram adicionados posteriormente visando um melhor funcionamento do sistema. Com isto, serão apresentados os requisitos funcionais onde serão esclarecidos ao pormenor todas as exigências técnicas que o sistema terá que possuir e os requisitos não-funcionais que, não acrescentando funcionalidades ao sistema, visam um melhor funcionamento e resposta do sistema a determinadas funções.

3.1.1 Requisitos funcionais

Como requisito funcional do sistema este deve apresentar um comportamento bem definido relativamente aos diferentes estados que o sistema terá que comportar em funcionamento, este requisito será explicado mais detalhadamente no subtema abaixo. Foi exigida configurabilidade por parte do utilizador no que toca à alteração dos valores de todas as variáveis parametrizadas do sistema.

3.2 Estados do sistema

Este requisito do sistema funcional incide, de uma forma genérica e pouco técnica, nos cenários de demonstração globais do sistema a desenvolver que nos foram exigidos. O comportamento deste sistema baseia-se em adquirir informação numa certa periodicidade dependendo do estado em que se encontra nesse dado instante. Esta periodicidade advém da necessidade de fazer com que o sistema consiga atuar mais vezes quando está em movimento e menos vezes quando imóvel, sendo que, no ponto de vista da monitorização, é muito mais valiosa a informação da *tag* quando esta está em movimento do que quando permanece imóvel.

Com isto foram propostos quatro cenários deveras distintos no ponto de vista do paciente:

Dispositivo estacionário		Dispositivo em movimento	
			
“IDLE”	“NO MOVEMENT”	“ACTIVE “	“MOVEMENT”

Tabela 4 - Estados do sistema

3.2.1 Requisitos não-funcionais

No que toca aos requisitos não-funcionais foi proposto a construção de um sistema o mais flexível possível onde se sobreponha ao máximo o cuidado pelos consumos energéticos reduzidos. Foi também mencionada a fácil configurabilidade do sistema que está assente sobre todas as variáveis necessárias para o bom funcionamento da máquina de estados designada.

3.3 Restrições

Este projeto teve algumas restrições particulares que devem ser especificadas. Em baixo vão ser apresentadas todas as restrições impostas que devem ser claras para que todo o avanço e progresso durante a implementação do sistema sejam claros e com menos dificuldades e redefinições possíveis.

3.3.1 Restrições técnicas

Desde início, foi colocada como principal restrição técnica o uso de um microcontrolador capaz de processar informação recebida pelo módulo sem-fios onde o microcontrolador terá que possuir memoria FLASH suficiente para suportar a aplicação final.

Também foi referido o facto de o processador *wireless* trabalhar com a especificação 802.11 b/g, visando assim uma maior flexibilidade de cobertura em termos de compatibilidades com outros pontos de acesso e comunicação. Por ultimo, um acelerómetro capaz de detetar movimentações na *tag* onde este terá um papel importante no que toca à comutação dos estados do sistema. Em suma teremos:

- Microcontrolador capaz de processar informação recebida pelo módulo
- Processador *wireless* IEEE 802.11 b/g
- Embutir um acelerómetro no sistema de maneira a detetar movimentações na *tag* e aumentar a periodicidade de varrimentos efetuados, obtendo assim menores consumos energéticos quando a *tag* não se encontra em movimento

3.3.2 Restrições temporais

Para além das restrições técnicas foram também impostas *deadlines* e objetivos a cumprir durante a execução deste projeto. Com isto, foi delimitado que este projeto teria que ser elaborado em catorze semanas e que as pesquisas iniciais relativas aos conceitos teóricos, documentadas no **Error! Reference source not found.**, teriam que ser apresentadas e discutidas com uma periodicidade de duas em duas semanas.

3.4 Descrição dos estados do sistema

Este diagrama de estados temporal foi inicialmente proposto e irá ser tido em questão quando o sistema estiver a ser implementado. Nesta imagem resume-se, muito sucintamente, todas as transições de estados bem como certos parâmetros e variáveis acima referidos.

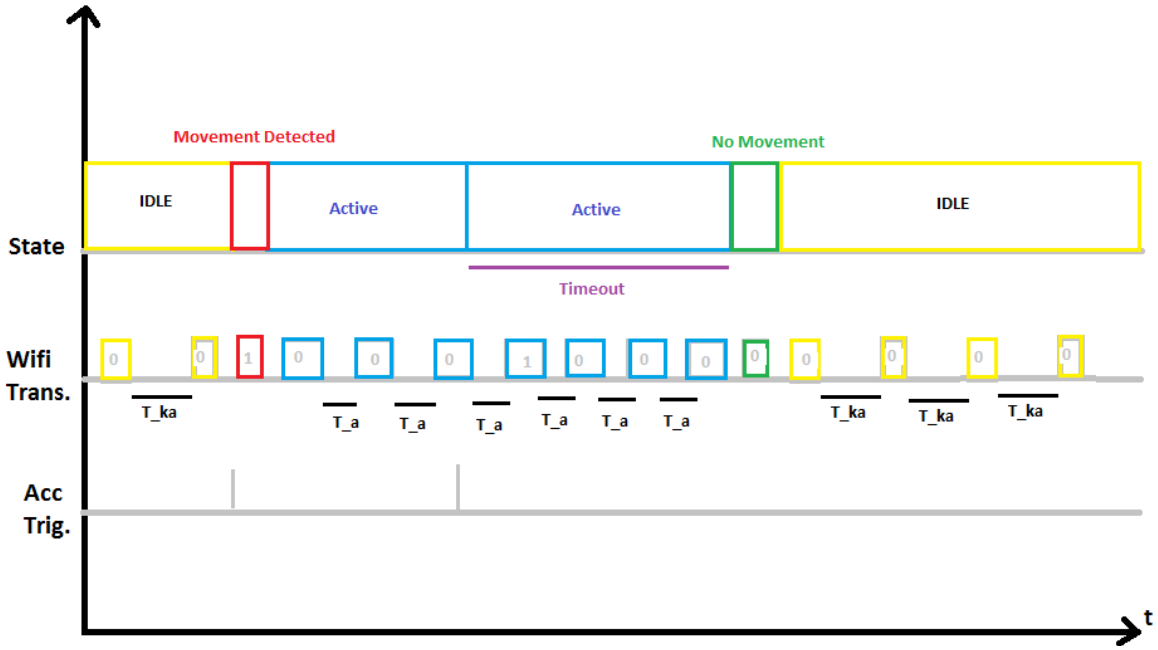


Figura 3.1 - Diagrama do estado em função do tempo com variáveis de entrada

Cenário em movimento – (IDLE)

Este estado em particular é ativado quando o estado anterior executado foi o estado "NO MOVEMENT".

O sistema permanece neste estado quando não é acionado movimento com ajuda de um acelerómetro ligado ao sistema.

Este sistema é responsável por:

- Procurar e enviar informação das redes wireless com o protocolo 802.11 que rodeiam o paciente com um intervalo definido pelo parâmetro IDLE_KEEPALIVE.
- Esperar por ocorrências de movimento com a ajuda do acelerómetro de maneira a transitar o estado do dispositivo para o "MOVEMENT DETECTED" com a finalidade de as leituras serem feitas com maior frequência, aumentando a amostragem do paciente quando este se movimenta.

Estado estacionário – (NO MOVEMENT)

O último estado integrado no grupo dos estados estacionários é ativado quando o estado anterior foi o "ACTIVE" e quando não houve nenhum acionamento por parte do acelerómetro durante o tempo máximo de espera.

O sistema permanece neste estado uma vez apenas em cada transição. Após a execução das tarefas abaixo indicadas o sistema transita automaticamente para o estado "IDLE" ou se

houver um disparo resultante da existência de movimento então o sistema transitará para o estado "ACTIVE" novamente.

Neste estado o sistema terá que ser responsável por:

- Procurar e enviar informação das redes wireless, com o protocolo 802.11 que rodeiam o paciente por um determinado número de vezes determinado pela variável N_MOVEMENTS.

Cenário em movimento – (ACTIVE)

Este segundo estado, referente à existência de movimento, é ativado quando o ultimo estado a ser executado foi o estado "MOVEMENT DETECTED" e a transição ocorre automaticamente sem outra condição adicional.

O sistema permanece neste estado enquanto os acionamentos de movimentos sejam intercetados pelo acelerómetro dentro de um tempo de espera parametrizado pela macro do pré processador TIMEOUT. Se esse tempo de espera for excedido e não tiver acontecido nenhum movimento então existe transição para o estado "NO MOVEMENT".

Neste estado o sistema terá que ser responsável por:

- Procurar e enviar informação das redes wireless com o protocolo 802.11 que rodeiam o paciente com um intervalo definido pelo parâmetro ACTIVE_TRANSMISSION.
- Intercetar acionamentos de movimento e reiniciar o temporizador, responsável pelo tempo de espera máximo (parametrizado pela variável TIMEOUT), quando estes aconteçam.

Cenário em movimento – (MOVEMENT DETECTED)

O estado "MOVEMENT DETECTED" é acionado assim que existe movimento e o acelerómetro é acionado.

O sistema permanece neste estado uma vez apenas em cada transição. Após a execução das tarefas abaixo indicadas o sistema transita automaticamente para o estado "ACTIVE".

Neste estado o sistema terá que ser responsável por:

- Procurar e enviar informação das redes wireless, com o protocolo 802.11 que rodeiam o paciente apenas por uma vez e em conjunto com estes dados enviar também a informação do disparo do acelerómetro (variável TRIGGER).

3.5 Diagrama de estados

Após todos os requisitos terem sido discriminados, chegou então a necessidade de elaborar um diagrama de estados do sistema que resumisse, solucionasse e simplificasse todas as questões existentes. A necessidade deste diagrama surge após os requisitos de todo o sistema terem sido apresentados. É então muito mais claro e sucinto desenhar um sistema com a máquina de estados tendo as propriedades e transições justificadas em baixo.

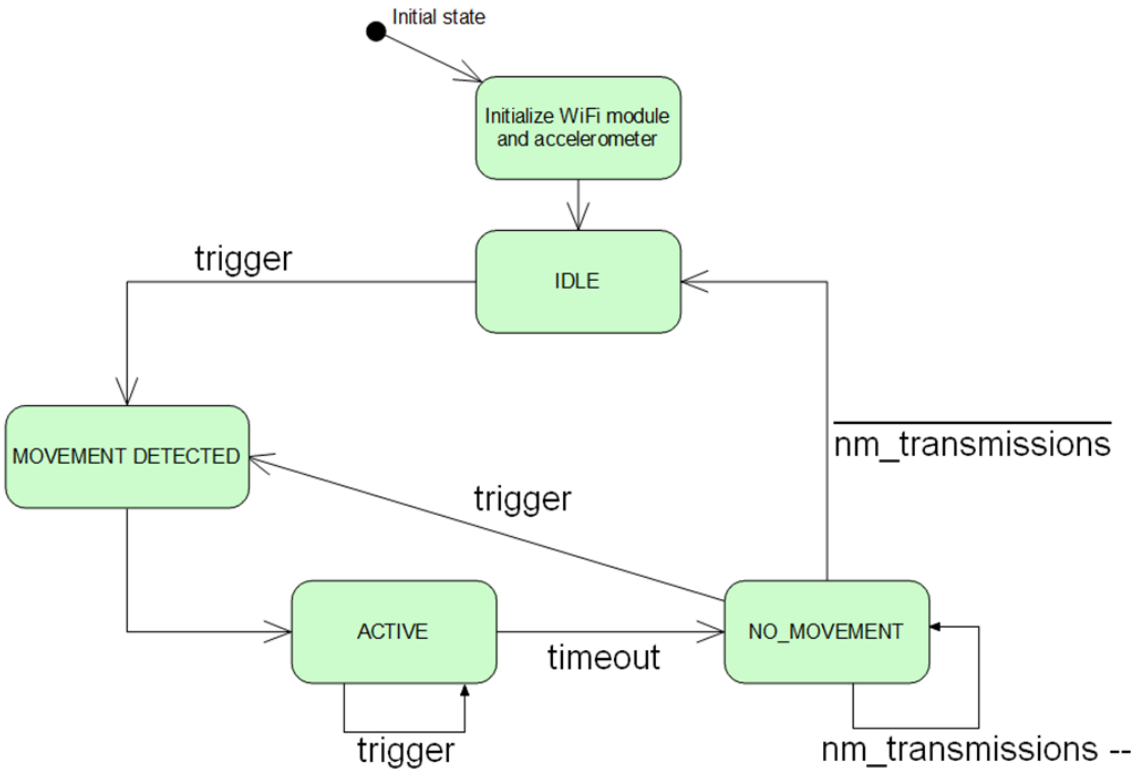


Figura 3.2 - Diagrama de estados do sistema e suas transições

3.6 Casos de uso

No diagrama abaixo é mostrado onde tanto o assistente como o operador da *tag* poderão interagir com o sistema. Assim, fixando o ponto-de-vista do assistente, este poderá, através de movimentos, acionar o acelerómetro e também ligar ou desligar o dispositivo se assim o pretender. Olhando agora para o operador, este poderá reconfigurar/reprogramar o dispositivo com os parâmetros escolhidos como também comutar a alimentação elétrica do dispositivo. Os parâmetros mencionados anteriormente são relativos aos temporizadores responsáveis pelas transições de estado da *tag*, servidor que aloja um interface preparado para receber a informação e finalmente o SSID e chave de segurança do ponto de acesso onde estará esse mesmo servidor.

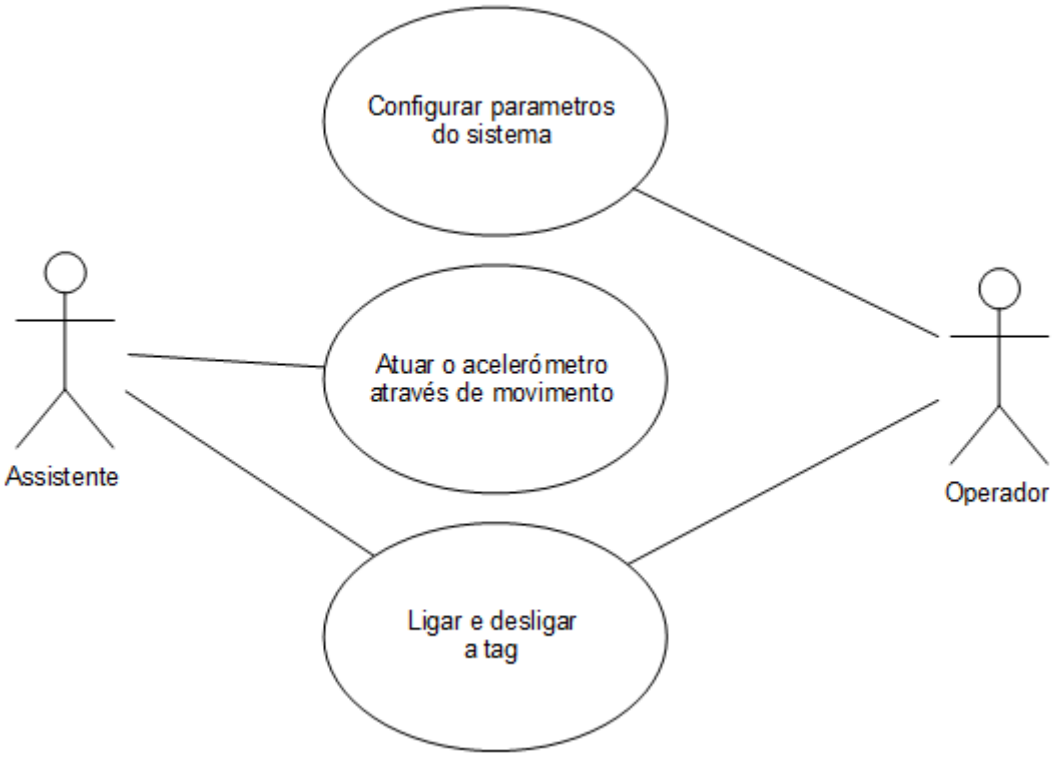


Figura 3.3 - Diagrama de casos de uso - Interação do assiste e operador sobre o sistema

3.7 Abordagem

Com esta análise conseguimos detetar a melhor abordagem a ter ao escolher o *hardware* a utilizar para completar todos os requisitos colocados. Assim deveremos começar por escolher um microcontrolador capaz de superar as exigências em termos de capacidade de processamento, um acelerómetro hábil para detetar acelerações criadas pelo movimento de um humano e um processador wireless IEEE 802.11 apto para analisar, conectar e enviar informação necessária.

Capítulo 4 *Design* do Sistema

Neste capítulo vamos analisar o *hardware* escolhido bem como as suas características gerais e particulares para que consigamos tirar um maior partido deste na fase da implementação. Sendo que o principal objetivo desta dissertação é estudar e implementar um software capaz de cumprir todos os objetivos iniciais propostos, teremos então que especificar também o software onde este vai ser construído, isto é, o IDE utilizado.

4.1 Especificações do hardware

Relativamente à escolha do *hardware*, foi proposta a placa de desenvolvimento WiFi DipCortex, desenhada e distribuída pela empresa SolderSpash Labs. Esta placa de desenvolvimento integra um processador ARM Cortex M3 manufaturado pela NXP e incluí também um módulo *wireless* da Texas Instruments SimpleLink CC3000MOD.

4.1.1 WiFi DipCortex

Com esta *board* conseguimos cumprir dois dos três requisitos impostos ao sistema que são o microcontrolador e o módulo Wi-Fi.

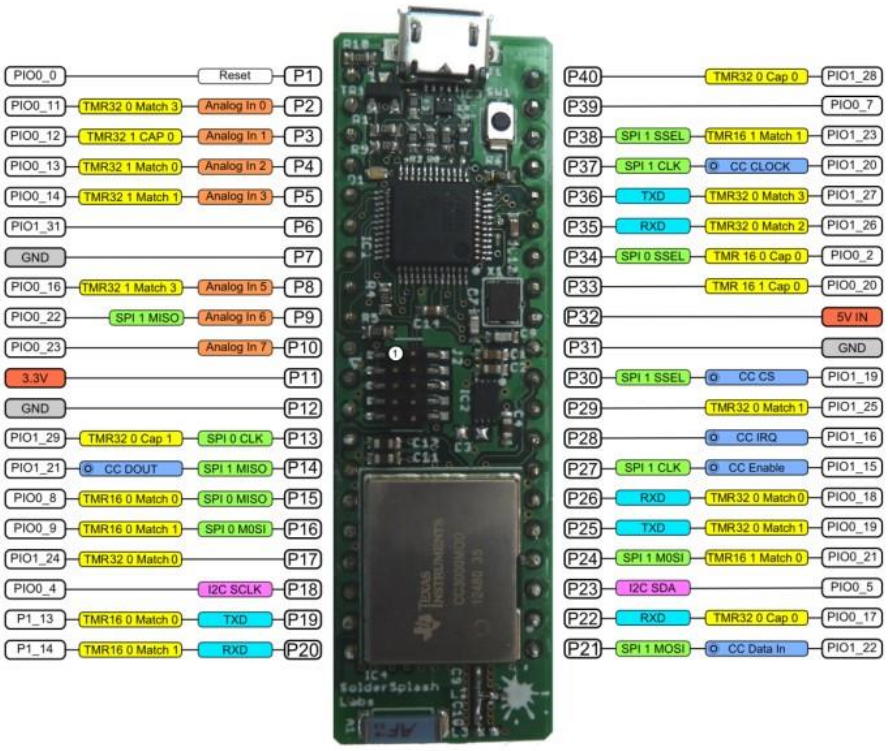


Figura 4.1 - Pin out da placa WiFi DipCortex e registos associados [30]

Relativamente ao microcontrolador este é o NXP ARM Cortex M3 LPC1347 com as seguintes especificações [27]:

- LPC1347 ARM Cortex M3
- Até 72MHz (frequência do oscilador)
- 64kB Flash
- 12kB SRAM
- 4kB EEPROM
- 12bit ADC – 7 canais
- USB 2 Full Speed – Com *stack* USB em ROM
- UART
- Barramento I2C
- Barramento SPI

No que toca ao módulo Wi-Fi teremos, como mencionado anteriormente, o processador TI SimpleLink CC3000MOD [28] WiFi Radio com as seguintes especificações [29]:

- 802.11b/g 2.4GHz
- WEP, WPA e WPA2
- *Open source* TI *control Stack*
- Configurável via *smartphone (*TI *smartconfig)*
- Incluída a Stack IPv4 TCP/IP em *hardware*
- *Low power shutdown*
- Armazenamento de perfis de rede, uso do modo *fast connection* em pontos de acesso já conhecidos
- DNS & mDNS *built in* [30]

Com esta escolha, onde são integrados estes dois módulos, conseguimos poupar tempo no que toca ao planeamento e impressão da placa e com isto aumentamos a produtividade, reduzindo o *time-to-market*. Também não pode ser esquecida a eficiência energética, pois ambos os componentes visam a sua integração em *designs low-cost* e *low-power*.

4.1.2 Acelerómetro

Relativamente ao acelerómetro, tratado pelo aluno Tiago Fernandes e abordado na sua dissertação de mestrado com o tema "*Server Application for Wi-Fi Tag Position Monitoring*", realizada na Universidade do Minho e datada de 2014, foi utilizado o módulo LIS331 manufaturado pela STMicroelectronics. Este acelerómetro foi desenhado para protótipos *low-power*, onde poderão ser obtidos valores axiais das três dimensões e alcançados deteções de orientações 6D. A sua *interface* de comunicação usa o protocolo SPI e este é utilizado para funções de deteção de movimento, deteção de queda-livre, monitorização de vibrações, reconhecimento de impactos, entre outros [31]. Posteriormente é explicada a sua integração por *software* no capítulo 5.3 Acelerómetro.

4.2 Especificações do *software*

O *software* responsável pelo desenvolvimento da aplicação ou *firmware* foi concebido pela empresa mbed. Esta dá suporte à placa de desenvolvimento WiFi DipCortex em termos das bibliotecas relativas ao HAL, drivers módulo *wireless* CC3000 e outras bibliotecas desenvolvidas pela comunidade mbed.

Esta empresa também disponibiliza uma plataforma onde está integrado um *browser* IDE onde todas as aplicações para as suas placas de desenvolvimento suportadas podem ser escritas, compiladas e carregadas para a mesma placa alvo. Neste IDE o utilizador tem acesso a um variado conjunto bibliotecas criadas pela comunidade mbed, como mencionada anteriormente, e pode também usufruir de um sistema de controlo de versões de *software* para que consiga estar sempre salvaguardado quando são feitas modificações e atualizações no decorrer da implementação.

No capítulo seguinte iremos abordar mais a pormenor o IDE *online*, um sistema de versões de *software* intitulado por "git" e uma pesquisa teórica realizada para aprimorar o conhecimento relativo à comunicação utilizada e seus protocolos. Com esta pesquisa consegue-se tirar um maior partido destas ferramentas quando a fase da implementação for iniciada.

4.2.1 Comportamento do sistema

A *tag* deve detetar quando está em movimento ou parada. Este movimento, após ter sido intercetado, serve para que nesse instante sejam coletados todos os dados das redes sem-fios com a especificação 802.11, também denominadas por Wi-Fi, que rodeiam a *tag*. Os dados adquiridos dos AP serão codificados com um determinado protocolo e enviados para um servidor alojado numa das redes já conhecidas. Os dados enviados de cada rede Wi-Fi serão:

- Nome do ponto de acesso (SSID)
- Endereço físico associado à interface de comunicação desse ponto de acesso (MAC *address* ou BSSID)
- Potência do sinal da rede em questão (RSSI)

Com esta informação também é enviado alguns dados relativos à *tag* e ao acelerómetro que a acompanha. Com isto também serão envidos:

- Endereço físico associado ao processador Wi-Fi da tag (*MAC address*)
- Valores axiais (X,Y,Z) presentes naquele dado instante pelo acelerómetro
- Estado em que se encontra o sistema naquele instante
- Variável booleana de disparo – caso o acelerómetro tenha sido acionado esta variável possuirá o valor "1".

Do lado do servidor, este deve receber a informação, enviada pela *tag* ou por um conjunto de *tags* em cada leitura, e monitorizá-la. Toda a parte de monitorização proposta irá ser realizada por outro aluno.

As imagens em baixo descrevem o comportamento do sistema nos variados estados assim haja a necessidade de se proceder a um varrimento e posteriormente o envio.

4.3 Estado IDLE

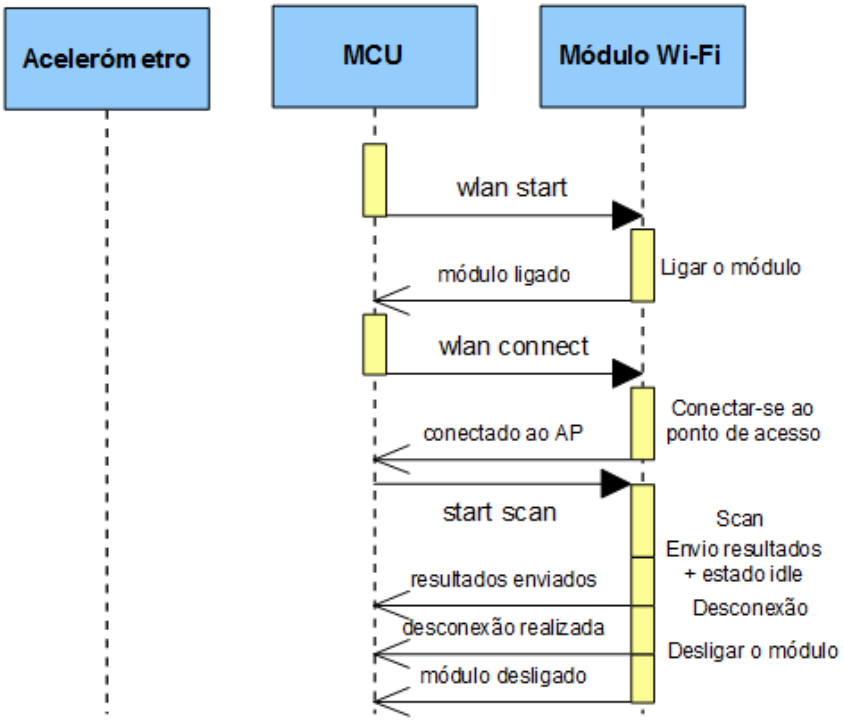


Figura 4.2 - Diagrama de sequência relativo ao estado IDLE

Com este diagrama de sequência fica então claro como é que o sistema se comporta neste estado ao enviar comandos para o módulo Wi-Fi. Foi então delimitado em ligar o módulo, proceder a todos os requisitos para enviar os resultados das redes 802.11, despachar o pacote de dados e no final desligar o módulo de maneira a reduzir o consumo energético. Sabendo que o temporizador responsável por realizar a sequência demonstrada na Figura 4.2 irá apontar para um valor de *overflow* perto dos vinte segundos, esta suposição faz com que a abordagem *turn-on turn-off* seja deveras vantajosa e faça, teoricamente, reduzir quatro vezes menos o consumo do sistema.

4.4 Estado MOVEMENT_DETECTED

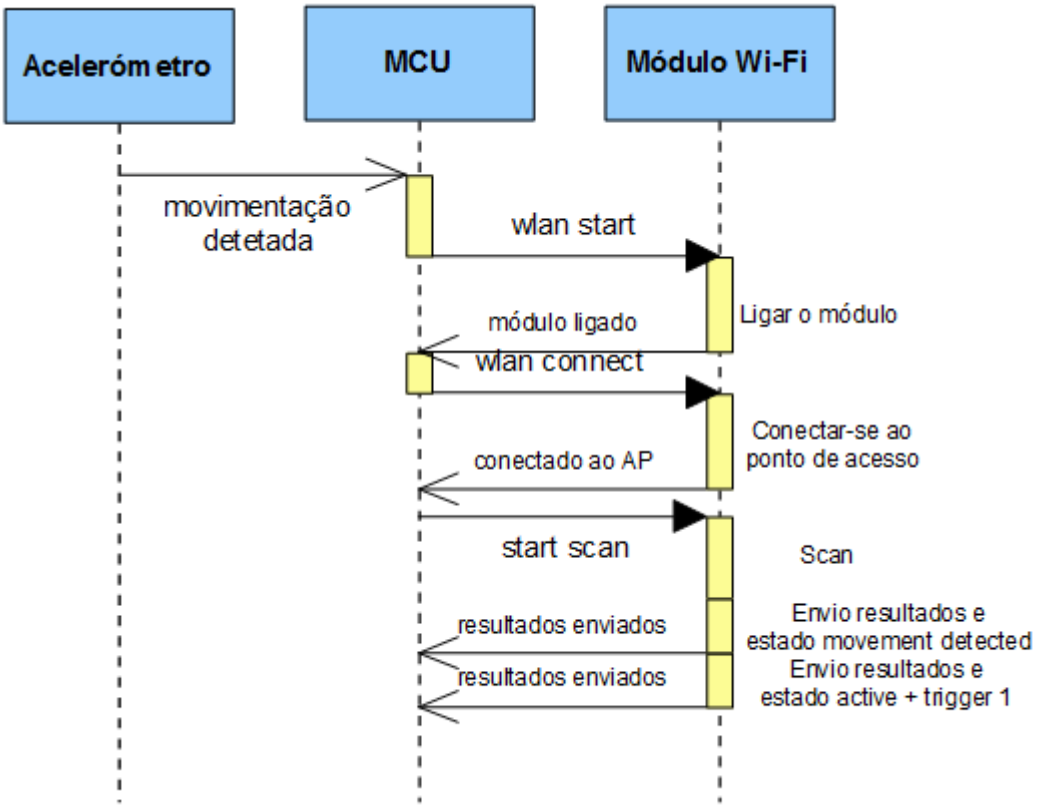


Figura 4.3 - Diagrama de sequência relativo ao estado MOVEMENT_DETECTED

O diagrama relativo ao estado *movement detected* está acima apresentado. Neste diagrama, após receber um *trigger* assíncrono por parte do acelerómetro, são enviados todos os comandos necessários, para o módulo TI CC3000, sendo possível realizar o restante processamento deste estado. Assim, depois de ligar e conectar o módulo a um ponto de acesso

são enviados dois pacotes, o primeiro contendo resultados do scan e o estado atual e o próximo contendo novamente os resultados mas desta vez incluindo estado em que o sistema irá transitar de seguida. Note que a informação da variável booleana *trigger* possui com o valor *true* derivado ao facto da transição para este estado ter sido realizada através de um acionamento de movimento.

4.5 Estado ACTIVE

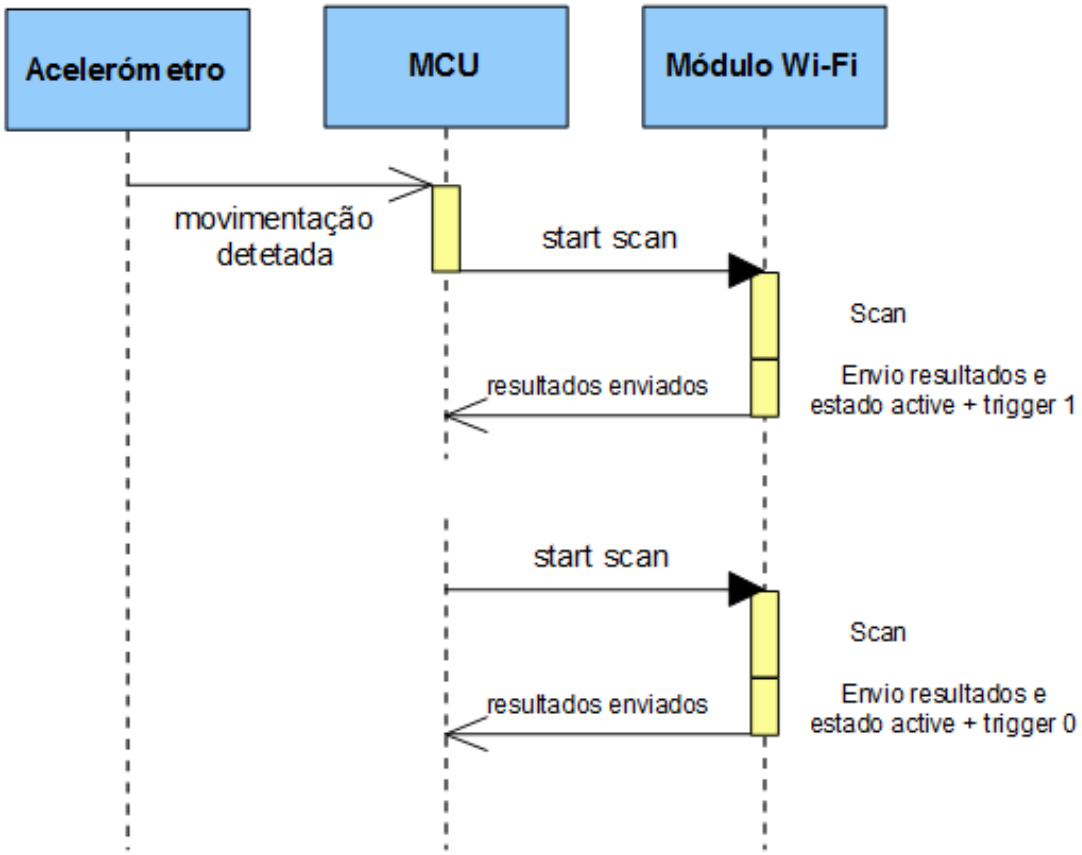


Figura 4.4 - Diagrama de sequência relativo ao estado ACTIVE

Neste exemplo verificamos que existem dois panoramas possíveis para que o sistema tenha que intervir, isto é, por acionamento de movimento e por temporizador. Sendo a fonte de disparo o único fator que diferencia as duas ações, temos então que diferencia-las posteriormente no envio pela variável *trigger* tal como acontece no estado anterior. Como já foi mencionado anteriormente, se neste estado não for recebida nenhuma ação despoletada por movimento então transitará para o estado caracterizado em baixo.

4.6 Estado NO_MOVEMENT

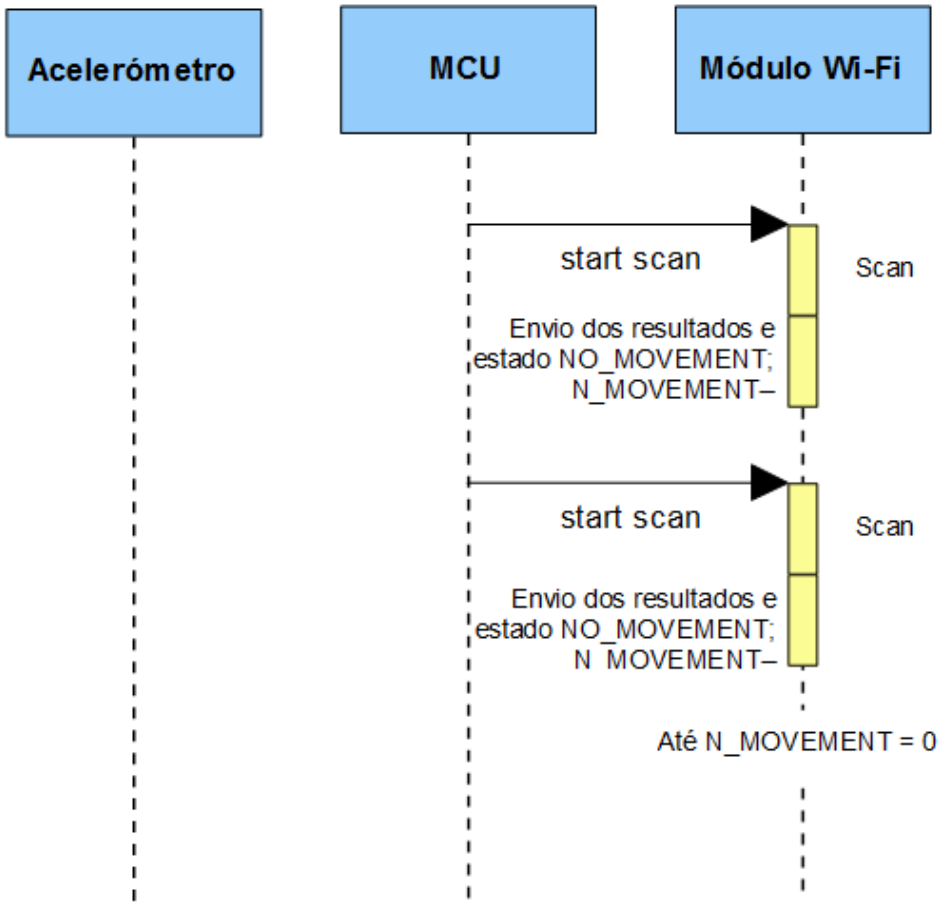


Figura 4.5 - Diagrama de sequência relativo ao estado NO_MOVEMENT

No último estado a ser caracterizado, conhecido como "NO_MOVEMENT", são enviados os resultados das informações recolhidas num determinado número de vezes. Esse número é parametrizado por uma macro *hard-coded* contido no ficheiro binário carregado na *flash* do microcontrolador. Entre os envios de informação, presentes neste estado, também é possível que o sistema seja interrompido por movimento transitando imediatamente para o estado "ACTIVE" com a variável *trigger* com *true*. Sem esta interrupção o sistema voltará ao estado "IDLE".

4.7 Casos de teste

Neste subtópico vão ser abordados vários testes que poderão ser realizados de maneira a validar o funcionamento do sistema. É deveras importante separar, testar e validar cada módulo para que consigamos encontrar erros ou *bugs* de uma forma mais rápida e concisa, perdendo assim menos tempo em *debug* de aplicações onde os módulos estão integrados.

Num ponto de vista generalizado do sistema podemos fazer com que este seja testado e dividido. Assim teremos:

- Envio pacote TCP/IP e UDP
- Scan das redes 802.11
- Leituras acelerómetro

4.8 *Overview* do sistema

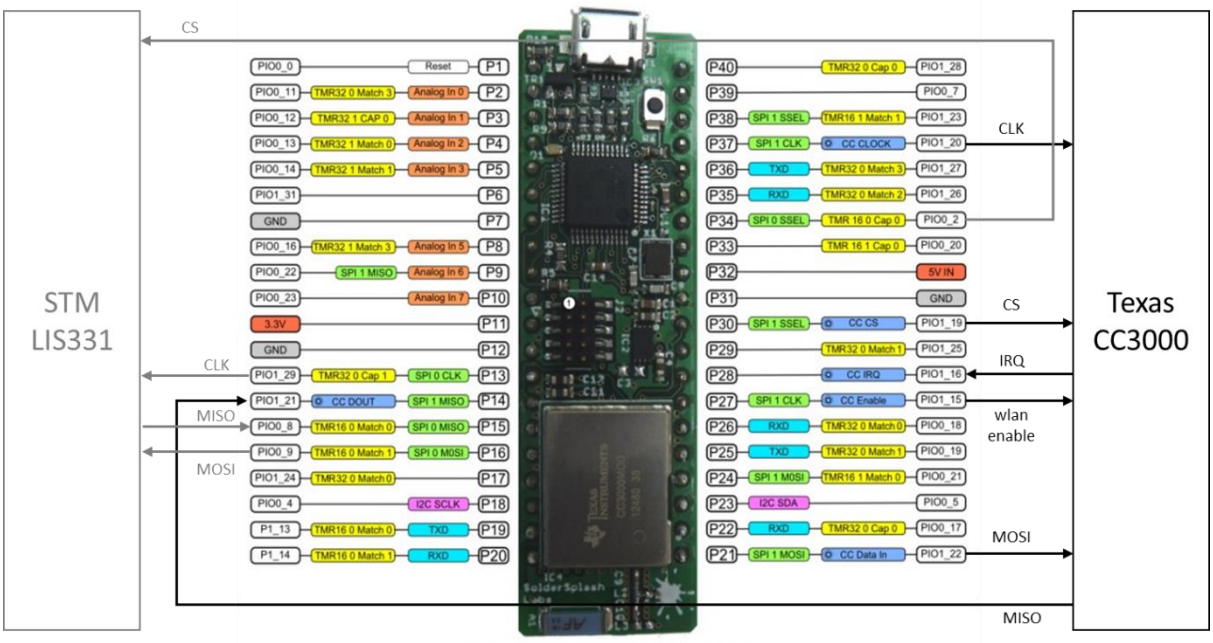


Figura 4.6 - Interfaces série e suas conexões entre os dispositivos e MCU

Na figura acima estão ilustradas as conexões físicas realizadas entre microcontrolador e os módulos, acelerómetro e *wireless*, que compõem o restante sistema. O protocolo de comunicação usado é o SPI que resulta num barramento série de quatro sinais lógicos: CLK, MOSI, MISO e CS. No caso do módulo 802.11, este integra um quinto sinal, IRQ, que faz com que quando o *slave* necessitar de enviar informação ao *master*, o canal IRQ gera uma interrupção ao MCU e assim este executa uma escrita *dummy* para conseguir ler o conteúdo de um determinado registo.

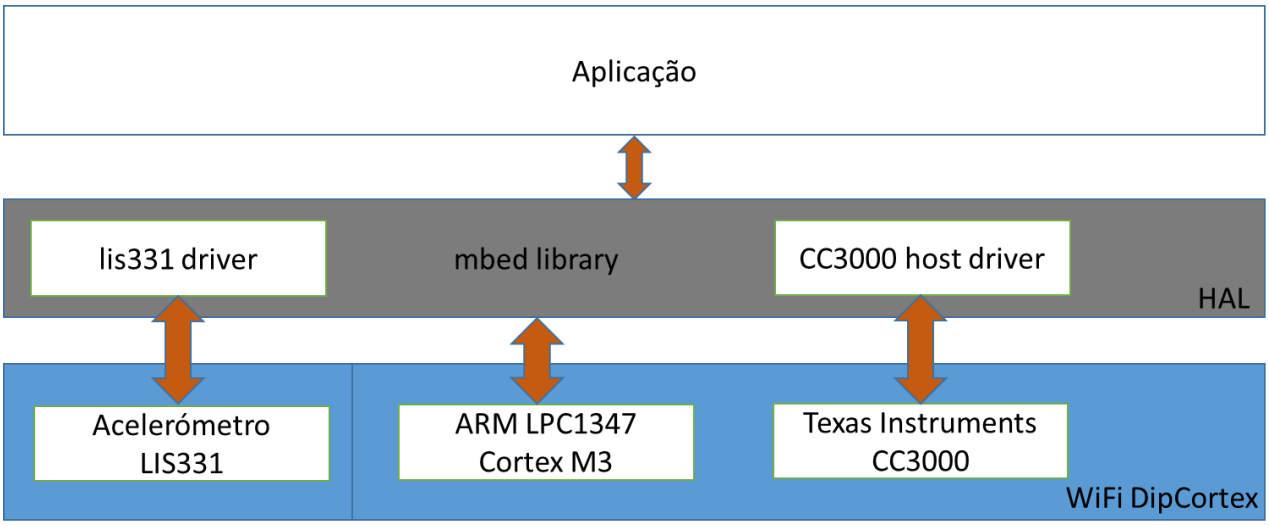


Figura 4.7 - *Overview* do sistema e dependência de níveis entre *hardware*, *drivers* e *software*

Neste *overview* do sistema conseguimos ter uma noção das dependências geradas pelo *hardware* e *software* presentes no projeto. Com o diagrama acima reparamos que existe uma forte relação das *drivers* Wi-Fi e do acelerómetro com o HAL, visto que as drivers necessitam diretamente de algumas APIs lá definidas. Contrariamente, o microcontrolador apenas necessita da HAL porque lá já residem os ficheiros referentes ao CMSIS Cortex-M3 Core Peripheral Access Layer e suas APIs.

Capítulo 5 Desenvolvimento do Sistema

Nesta fase, onde toda a teoria já foi estudada e documentada, a análise e *design* do sistema também detalhados, a tarefa é colocar em prática tudo o que foi exposto até agora.

A componente prática é o que realmente resulta do estudo prévio e esta denota uma importância extrema em projetos como este onde a funcionalidade é a prioridade. Posto isto, irão ser explicadas todas as decisões baseadas em conceitos teóricos assimilados, implementações e alguns diagramas demonstrativos, facilitando assim a compreensão por parte do leitor.

Este tópico será iniciado com implementação relativa aos casos de teste explicados em 4.3 Casos de teste. Numa fase posterior será demonstrada a implementação relativa ao protocolo de envio e seu encapsulamento seguido com a explicação relativa à decisão das escolhas tomadas. Seguidamente será abordada toda a construção da máquina de estados que dará suporte à comutação de estados do sistema e finalmente a integração do acelerómetro e teste final sobre o sistema unitário.

5.1 Organização da árvore de ficheiros

A disposição dos ficheiros contendo código-fonte é determinante para uma implementação organizada. Neste caso foi decido implementar uma solução modular onde são integrados vários ficheiros respeitantes a implementações dos diversos módulos do sistema. Posto isto, na figura abaixo é exibida a disposição dos ficheiros no IDE utilizado.

	cc3000_hostdriver_mbedsocket		Published Library	2 weeks, 2 days ago
	lis331.cpp	10.2 kB	C/C++ Source File	3 weeks, 3 days ago
	lis331.h	1.0 kB	C/C++ Header File	19 Jul 2014
	main.cpp	8.1 kB	C/C++ Source File	1 week, 1 day ago
	main.h	1.2 kB	C/C++ Header File	2 weeks, 2 days ago
	wifi.h	1.1 kB	C/C++ Header File	1 week, 1 day ago
	mbed		Library Build	2 weeks, 2 days ago

Figura 5.1 - *File system* do projeto

Temos então, a nível da implementação, uma solução desenvolvida em linguagem C++ onde os ficheiros:

- lis331.cpp e lis331.h, *source* e *header* respetivamente, correspondem à *driver* do acelerómetro
- wifi.h que contem a definição da estrutura de *scan* e macros relativas ao ponto de acesso onde será realizado o *broadcast* da informação
- main.h incorpora uma serie de definições de funções, implementadas no ficheiro main.cpp, e a parametrização dos tempos que vão servir para *compare match* nos temporizadores
- mbed e cc3000_hostdriver_mbedsocket são duas bibliotecas importadas no IDE mbed que foram desenvolvidas não só pelos programadores oficiais desta empresa como também pela comunidade

5.2 Temporizadores

Com a inclusão da biblioteca mbed no projeto, faz com que seja possível a integração e uso de certas APIs já implementadas. O uso dos temporizadores implementados nesta biblioteca é um dos casos mais notáveis deste sistema. Esta implementação, realizada pela equipa da mbed, inclui *timers* em *software* onde são incrementados via systick fornecido pelo oscilador do microcontrolador.

Estes três timers estão associados diretamente a dois estados, sendo que o primeiro, denominado por T_ka, mais precisamente *keep_alive* associa-se ao estado IDLE, T_active e T_timeout agrupados ao estado ACTIVE.

```
/* ---------------- TIMERS ---------------- */
Timer T_ka;
Timer T_active;
Timer T_timeout;
```

Figura 5.2 - Temporizadores presentes no sistema

Para estes temporizadores existem também certos parâmetros definidos, como macros do pré-processador, que assistem à comparação do valor atual do timer quando este está a ser incrementado.

```
#define IDLE_KEEPALIVE 15
#define ACTIVE_TRANSMISSION 5
#define TIMEOUT 60
#define N_MOVEMENTS 5
```

Figura 5.3 - Macros do pré-processador associados aos temporizadores

5.3 Acelerómetro

5.3.1 Configuração

Relativamente ao driver usado para comunicar com o acelerómetro, este foi cedida pelo ESRG, onde já se tinham elaborado projetos com o mesmo circuito integrado. Com isto, tivemos apenas que configurar as opções necessárias de modo a que o acelerómetro apenas interviesse na gama de movimentos necessária.

Foi então configurado com um *threshold* e mantido o valor de *full-scale selection* para acelerações até 2G. Relativamente à fonte de interrupção, este foi configurado para ser acionado pelo apenas pelo eixo Z, relativo ao movimento de deslocação horizontal, partindo do principio que a *tag* está paralela ao corpo de um humano. Os outros dois eixos, X e Y, estão ativos mas apenas para leituras, pois irão ser necessários para que sejam lidos os seus valores e enviados sempre que ocorre a necessidade de um *scan*.

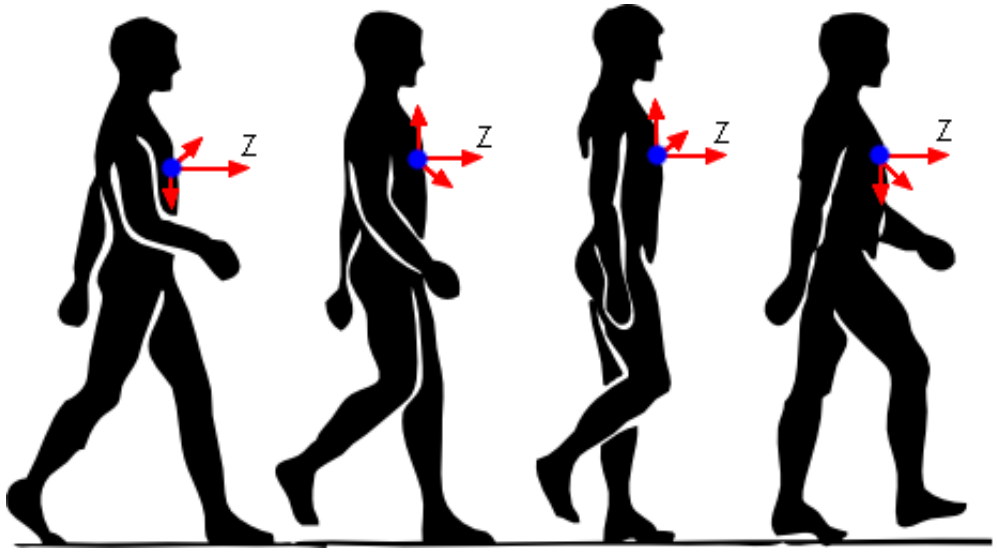


Figura 5.4 - Acelerómetro presente no caso de uso [37]

Com a imagem em cima conseguimos entender que, assegurando que o módulo está paralelo ao corpo, o eixo do Z terá sempre a mesma direção. Os eixos negativos dos eixos respetivos não estão representados na imagem porque todas as movimentações detetadas são resultado do módulo matemático do vetor resultante nesse eixo fazendo que todos os valores sejam positivos.

```
lis331_read_reg(0);

if(lis331_read_reg(LR_WHO_AM_I) == I_AM_ADDRESS)
{
    lis331_write_reg(LR_CTRL_REG1, LR_POWER_NORM | LR_DATA_RATE_50 | LR_Z_ENABLE | LR_Y_ENABLE | LR_X_ENABLE);
    lis331_write_reg(LR_CTRL_REG2, 0x00);
    lis331_write_reg(LR_CTRL_REG3, 0x00);
    lis331_write_reg(LR_CTRL_REG4, LR_FS_2G);
    lis331_write_reg(LR_REFERENCE, 0x00);
    lis331_write_reg(LR_INT1_THS, THRESHOLD);
    lis331_write_reg(LR_INT1_DURATION, 0x00);
    lis331_write_reg(LR_INT2_THS, 0x00);
    lis331_write_reg(LR_INT2_DURATION, 0x00);
    lis331_read_reg(LR_HP_FILTER_RESET);
    lis331_write_reg(LR_INT1_CFG, LR_INT1_SOURCE_6MOV | LR_INT1_ZHIE_ENABLE | LR_INT1_ZLIE_ENABLE);
    lis331_write_reg(LR_INT2_CFG, 0x00);
    lis331_write_reg(LR_CTRL_REG5, LR_TURNON_SLEEP);
```

Figura 5.5 - Método responsável pela configuração do acelerómetro

Na Figura 5.5 está implementada a função de configuração onde é inicialmente verificado se o endereço do dispositivo que se está a comunicar coincide com o dispositivo que se pretende configurar. De seguida são carregados os valores pretendidos para os registos de configuração e controlo. Ambos os registos e alguns valores estão definidos com macros facilitando assim a configuração ao programador.

Como podemos reparar nos valores de cada registo, temos então no registo CTRL_REG4 a configuração *full-scale* para 2G e o valor de *threshold* presente no registo LR_IN1_THS. No que toca aos eixos, temos no registo CTRL_REG1 os três eixos ligados mas apenas um deles está conectado à interrupção INT1 presente no registo LR_INT1_CFG com os valores LR_INT1_ZHIE_ENABLE e LR_INT1_ZLIE_ENABLE.

5.3.2 Integração

Para que este driver fosse conectado com sucesso foram então adicionados dois membros privados à classe.

```
class LIS331
{//atributes
    private:
        SPI* _spi;
        DigitalOut* _cs;
//methods
    public:
            LIS331();
            ~LIS331();
    void    lis331_init();
    void    lis331_init(uint8_t n_bits, uint8_t mode, uint8_t freq);
    bool    lis331_config();
    void    lis331_write_reg(uint8_t reg, uint8_t value);
    uint8_t   lis331_read_reg(uint8_t reg);
    uint8_t  lis331_getXValue();
    uint8_t   lis331_getYValue();
    uint8_t   lis331_getZValue();
};
```

Figura 5.6 - Acelerómetro - *class* LIS331

Estes dois membros estão definidos na biblioteca mbed que implementa o HAL do microcontrolador. Assim, sempre que esta classe é instanciada o seu construtor reservará memória para dois objetos dos tipos SPI e DigitalOut.

```
LIS331::LIS331()
{
    this->_spi = new SPI(p16, p15, p13);
    this->_cs = new DigitalOut(p34);
}
```

Figura 5.7 – Acelerómetro - Junção da *class* LIS331 com a biblioteca mbed

Os argumentos, presentes no construtor SPI, sendo p16, p15 e p13 são respetivamente SPI0_MOSI, SPI0_MISO e SPI0_CLK. Com isto fazemos a associação entre o *pinout* relativo ao SPI0 com o driver cedido.

No que toca ao bit CS, é instanciado um objeto do tipo DigitalOut e dado como argumento o p34. Este bit é internamente comutado nas funções de envio da biblioteca do acelerómetro que irá ser mostrado mais a baixo.

```
void LIS331::lis331_init()
{
    this->_spi->format(8,3);
    this->_spi->frequency(1000000);
}

void LIS331::lis331_init(uint8_t n_bits, uint8_t mode, uint8_t freq)
{
    this->_spi->format(n_bits, mode);
    this->_spi->frequency(freq);
}
```

Figura 5.8 – Acelerómetro - *class* LIS331 - Inicialização

Nesta imagem é demonstrado o método de inicialização do acelerómetro. Após o objeto _spi ter sido instanciado no seu construtor vazio, podemos então usar as APIs fornecidas pela mbed para definir o número de bits por trama, modo de operação e a frequência de comunicação. Este método lis331_init() conta com dois *overloads* em que no primeiro não existem argumentos e as definições são forçadas a 8 bits de dados, modo 3 de operação e frequência de 1Mhz ou então o segundo *overload* deixa ao critério do programador definir o numero de bits por trama, modo de operação e frequência de comunicação.

```
void LIS331::lis331_write_reg(uint8_t reg, uint8_t value)
{
    reg = reg & SPI_W;

    *this->_cs = 0;
    this->_spi->write(reg);
    this->_spi->write(value);
    *this->_cs = 1;
}

uint8_t LIS331::lis331_read_reg(uint8_t reg)
{
    uint8_t value;

    reg = reg | SPI_R;

    *this->_cs = 0;
    this->_spi->write(reg);
    value = this->_spi->write(0);
    *this->_cs = 1;

    return value;
}
```

Figura 5.9 - Acelerómetro - Função de leitura e escrita

Relativamente ao envio e receção, no método lis331_write_reg verificamos o envio da trama composta pelo registo seguidamente do valor a carregar nesse registo. Já no segundo método, lis331_read_reg, como retorno da função temos o valor a receber e como argumento o endereço que pretendemos ler. Note que o bit CS é comutado por *software* a cada vez que é executada uma leitura ou uma escrita. Com esta informação relativa ao modo como são lidos os registos que o utilizador pretende, conseguimos assim demonstrar como são adquiridos os valores das três dimensões presentes no acelerómetro.

```
uint8_t LIS331::lis331_getXValue(void)
{
  return (lis331_read_reg(LR_OUT_X_H));
}

uint8_t LIS331::lis331_getYValue(void)
{
  return (lis331_read_reg(LR_OUT_Y_H));
}

uint8_t LIS331::lis331_getZValue(void)
{
  return (lis331_read_reg(LR_OUT_Z_H));
}
```

Figura 5.10 - Acelerómetro - Leitura dos valores axiais

Foram demonstrados todos os métodos para configuração, escrita e leitura de valores e registos. Na figura abaixo exibimos o objeto do tipo InterrupIn, presente na biblioteca mbed, que recebe como argumento o *pin* responsável pela fonte de interrupção. Já o objeto lis331 invoca automaticamente o construtor vazio LIS331::LIS331() que por sua vez inicializa o protocolo SPI e o bit CS, explicado no início deste capitulo.

```
/* ---------------- ACCELEROMETER ---------------- */
InterruptIn int_acc(p6);
LIS331* lis331 = new LIS331();
```

Figura 5.11 - Acelerómetro - Inicialização dos objetos

Com isto foi criada uma função onde não só são chamadas as funções explicadas anteriormente, tais como list331_init() e lis331_config(), mas também as funções que configuram o acionamento da interrupção em *rising edge* e a função responsável pelo *handling* da interrupção.

```
void config_and_start_acc( void )
{
    lis331->lis331_init();
    lis331->lis331_config();
    int_acc.rise(&funcao);
    int_acc.enable_irq();
}
```

Figura 5.12 - Acelerómetro - Integração e configuração no sistema

Após a função config_and_start_acc() ser chamada na aplicação, conseguimos assim ter acesso às deteções de movimento geradas pelo acelerómetro sendo produzida a invocação da função função(), visto que esta foi passada como argumento em int_acc.rise($funcao). A chamada int_acc_enable_irq() representa a habilitação do salto ao *handler* da interrupção. (Consultar 5.6.5 Handler da interrupção)

5.4 Comunicação com o módulo CC3000

5.4.1 Inicialização do CC3000

O módulo CC3000 requer um certo cuidado com a inicialização dos seus objetos. Este procedimento terá que ser realizado nesta ordem visto que, por exemplo, o objeto wifi do tipo cc3000 estará internamente conectado com o objeto server do tipo Endpoint. Esta ligação é produzida na biblioteca que dá suporte ao CC3000 com a biblioteca que implementa os *sockets* fazendo que para se inicializar um Endpoint temos que garantir que já existe um objecto cc3000.

```
/* ---------------- WI FI ---------------- */
using namespace mbed_cc3000;
cc3000 wifi(p28, p27, p30, SPI(p21, p14, p37));
Endpoint server;
```

Figura 5.13 - Inicialização CC3000

5.4.2 Scan

A função responsável por efetuar o varrimento das redes 802.11 foi implementada recorrendo a APIs e estruturas já disponíveis, documentadas no *datasheet* do módulo CC3000. Nesta implementação teve-se apenas o cuidado em tratar essas APIs, certificando que o uso das mesmas estava a ser feito da forma mais otimizada.

As APIs mais importantes são:

- wlan_ioctl_set_scan_params – onde são definidos os parâmetros do varrimento e o seu inicio

- wlan_ioctl_get_scan_results – onde são recebidos os resultados dos varrimentos e carregados para uma estrutura previamente criada

Com a ajuda da estrutura implementada conseguimos ter acesso às opções de varrimento do módulo WiFi, iniciar a procura e obter os resultados recebidos nesse mesmo objeto.

```
typedef struct Result_Struct
{
  uint32_t  num_networks;
  uint32_t  scan_status;
  uint8_t   valid:1;
  uint8_t   rssiByte:7;
  uint8_t   Sec_ssidLen;
  uint16_t  time;
  uint8_t   ssid_name[32];
  uint8_t   bssid[6];
} ResultStruct_t;
```

Figura 5.14 – CC3000 – Estrutura de dados presente em cada *scan*

Como já foi mencionado, os valores necessários presentes nesta estrutura para posterior envio serão: SSID (Nome do ponto de acesso), RSSI (Potencia de sinal) e BSSID (Endereço fisico MAC do ponto de acesso).

Já num contexto de execução, a abordagem realizada para que fosse possível popular a estrutura presente na Figura 5.14 será demonstrada na Figura 5.15.

Com isto foram implementadas as seguintes funções:

```
void Wifi_StartScan ( uint32_t enable )
{
    const uint32_t uiMinDwellTime = 20;
    const uint32_t uiMaxDwellTime = 30;
    const uint32_t uiNumOfProbeRequests = 2;
    const uint32_t uiChannelMask = 0x1fff;
    const int32_t iRSSIThreshold = -80;
    const uint32_t uiSNRThreshold = 0;
    const uint32_t uiDefaultTxPower = 205;
    const uint32_t aiIntervalList[16] = { 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000, 2000,
    2000, 2000, 2000, 2000, 2000, 2000, 2000 };

    wifi._wlan.ioctl_set_scan_params(enable, uiMinDwellTime, uiMaxDwellTime, uiNumOfProbeRequests, uiChannelMask,
    iRSSIThreshold, uiSNRThreshold, uiDefaultTxPower, (uint32_t *)aiIntervalList);
}



void Wifi_GetScanResults ( void )
{
    uint8_t i = 0;
    uint8_t encryptionType = 0;
    uint8_t ssidLen = 0;
    ResultStruct_t scanResults;
    int8_t tempRssi = 0;

    Wifi_StartScan(1);

    wifi._wlan.ioctl_get_scan_results(0, (uint8_t*)&scanResults);
    pc.printf("\r\n Scan found %u WiFi networks", scanResults.num_networks);

    i = scanResults.num_networks;
    while (i) {
        if (scanResults.valid) {
            encryptionType = (scanResults.Sec_ssidLen & 0x03);

            ssidLen = scanResults.Sec_ssidLen >> 2;
            if ( ssidLen < 32 ) {
                scanResults.ssid_name[ ssidLen ] = 0;
            }

            tempRssi = (0x80 | scanResults.rssiByte);
            pc.printf("\r\n %s (%s) RSSI (%d, %d dBm)", scanResults.ssid_name, WIFI_SEC_TYPE[encryptionType],
            scanResults.rssiByte, tempRssi);
        }
        wifi._wlan.ioctl_get_scan_results(0, (uint8_t*)&scanResults);

        i --;
    }
    Wifi_StartScan(0);
}
```

Figura 5.15 – CC3000 – Configuração, início da pesquisa, armazenamento dos dados e interrupção da

Os valores presentes na função de configuração são os recomendados pela Texas Instruments.

5.4.3 Conectar a um AP

Nesta implementação, foi decidido criar uma função responsável por ligar com a conexão. Em baixo é mostrada a sua implementação.

```
bool configure_and_connect ( void )
{
    if(wifi.is_connected()) return true;
    Timer t;
    wifi.start(0);
    wait_ms(750);
    wifi.connect_non_blocking((uint8_t *)SSID, (uint8_t *)AP_KEY, AP_SECURITY);

    t.start();
    while(!wifi.is_connected()) {
        if (t.read_ms() > 10000) {
            return false;
        }
    }
    return true;
}
```

Figura 5.16 - CC3000 - Conexão a um AP

Assim, depois da chamada `wifi.start(0)` e o *delay* que deve ser cumprido para que o módulo seja ligado, é feito o pedido de ligação à rede. Foi engendrado um timer de modo a que se a conexão não fosse realizada dentro de um tempo predefinido, neste caso cem mil milissegundos = dez segundos, esta mesma função retornava o valor *false*. Os parâmetros SSID, AP_KEY e AP_SECURITY estão definidos como:

```
// use this defines in AP_SECURITY
#define NONE 0
#define WEP  1
#define WPA  2
#define WPA2 3

 // Default SSID Settings
#define SSID            "AndroidAP"
#define AP_KEY          "casalage"
#define AP_SECURITY     WPA2
```

Figura 5.17 – CC3000 - Parâmetros relativos à conexão presentes em wifi.h

5.4.4 Envio de dados

Relativamente ao envio de dados, mais precisamente sobre o protocolo de comunicação, teria que ser escolhido o encapsulamento que melhor correspondesse aos requisitos do sistema. Posto isto, o encapsulamento que responderia melhor às necessidades exigidas foi o protocolo UDP pela velocidade de transmissão comparativamente com o encapsulamento TCP aliada com o facto de não serem usados ACKs, possibilidade de uso de *broadcast* e pela pouca relevância dada ao facto da perda de alguns dados entre a comunicação. Ver 2.6.4.1 TCP e 2.6.4.2 UDP.

```
/* ---------------- UDP buffer ---------------- */
char tmpBuffer[600] __attribute__((section("AHBSRAM0")));

void send_udp_frame ( void )
{
    if(wifi.is_connected()) {
        UDPSocket UDP_TransmitSocket;
        UDP_TransmitSocket.set_blocking(false, 2000);       //TI nao assegura isto
        if(!UDP_TransmitSocket.bind(1900)) {
            UDP_TransmitSocket.sendTo(server, tmpBuffer, strlen(tmpBuffer));
        }
    }
}
```

Figura 5.18 - CC3000 – Buffer de dados e função de envio

No exemplo acima foi reservado espaço para 600 bytes na RAM0. Este *buffer* serve para armazenar os dados relativos a um *scan* para posteriormente ser despachado pela função send_udp_frame(). Nesta função é verificado se o módulo CC3000 está associado ao ponto de acesso predefinido e se esta condição se verificar é criado o objeto do tipo UDPSocket, inicializado e constituído o *bind* a uma *source port* definida (opcional) e finalmente é enviado o buffer juntamente com o objeto server presente na Figura 5.13 explicado na Figura 5.22.

5.5 Notação atribuída – JSON

No que toca ao formado de dados presente nos pacotes de varrimentos, foi aconselhado que estes possuíssem uma notação uniformizada de modo a que o intercâmbio de dados computacionais fosse o mais leve e genérico possível. Assim, surgiu a ideia de utilizar o formato JSON. Este formato surgiu, inicialmente na linguagem JavaScript, fazendo face ao conhecido XML, mas este está cada vez mais popular noutras linguagens com o aparecimento de *deserializers* criados por terceiros. Neste momento, este formato de dados não só está presente na linguagem de programação JS mas também em C/C++, C#, PERL, Python, PHP, etc.

Contextualizando e incluindo esta decisão para o sistema alvo, teremos do lado da *tag* os varrimentos formatados na sintaxe JSON e do lado do servidor haverá um analisador separando e restruturando os dados recebidos através de um JSON *deserializer*.

A formatação JSON acompanha a seguinte sintaxe:

```
{
      "TYPE":"MOVEMENT_DETECTED",
```

```
        "TAG_MAC":"08:00:28:57:43:d4",
        "AP_READING": [
              {"MAC":"00:1f:9e:cd:bb:b0","SSID":"CCG_AD","RSSI":"-39"},
              {"MAC":"00:1f:9e:cd:bb:b2","SSID":"guest","RSSI":"-89"},
              {"MAC":"00:1f:9e:cd:bb:b1","SSID":"","RSSI":"-58"},
              {"MAC":"00:1f:9e:80:ce:80","SSID":"eduroam","RSSI":"-78"}
        ],
        "ACCELEROMETER":{"TRIGGER":"1","X":"1","Y":"255","Z":"16"}
}
```

Com este formato reparamos que existem quatro parâmetros em cada pacote de varrimento enviado. Os parâmetros são: "TYPE" representa o estado atual em que o sistema se encontra, "TAG_MAC" identifica o endereço físico da tag que envia a informação, "AP_READING" exibe uma lista de pontos de acesso capturados em que cada um desses resultados está contido o BSSID,SSID e RSSI, finalmente temos o parâmetro "ACCELEROMETER" que contem a variável já referida como *trigger* e as inclinações axiais das três dimensões adquiridas no momento da captura.

```
char udp_frame_scan[200];
sprintf(udp_frame_scan,"{\"MAC\":\"%02x:%02x:%02x:%02x:%02x:%02x\",\"SSID\":\"%s\",\"RSSI\":\"%d\"}",
scanResults.bssid[0], scanResults.bssid[1], scanResults.bssid[2], scanResults.bssid[3], scanResults.bssid[4],
scanResults.bssid[5],scanResults.ssid_name,tempRssi);
str_udp_frame_scan += string(udp_frame_scan);

uint8_t myMAC[8];
wifi.get_mac_address(myMAC);
char udp_frame_mac[200];
sprintf(udp_frame_mac,"{\"TYPE\":\"%s\",\"TAG_MAC\":\"%02x:%02x:%02x:%02x:%02x:%02x\",\"AP_READING\":[",type.c_str(),
myMAC[0], myMAC[1], myMAC[2], myMAC[3], myMAC[4], myMAC[5]);


char udp_frame_acc[200];
sprintf(udp_frame_acc,"\"ACCELEROMETER\":{\"TRIGGER\":\"%d\",\"X\":\"%d\",\"Y\":\"%d\",\"Z\":\"%d\"}}",trigger,lis331->
lis331_getXValue(),lis331->lis331_getYValue(),lis331->lis331_getZValue());

memset(tmpBuffer, '\0', sizeof(tmpBuffer));
sprintf(tmpBuffer, "%s%s],%s",udp_frame_mac, str_udp_frame_scan.c_str(), udp_frame_acc);

send_udp_frame();
```

Figura 5.19 - Formatação JSON na aplicação

Na Figura 5.19 são formatadas, em sintax JSON, as informações de cada leitura em *sub strings* antes de se proceder ao envio. Existem três *strings* secundárias: udp_frame_mac[], udp_frame_scan[] e udp_frame_acc[] acolhem, respetivamente, os dados relativos à *tag* e estado atual do sistema; *scan* das redes 802.11 vizinhas; valores axiais do acelerómetro e campo TRIGGER. Para que estas *sub strings*, já formatadas em JSON, sejam encapsuladas num só

buffer são realiadas as funções memset e seguidamente o sprintf. A inicialização, explicação referente ao *buffer* tmpBuffer e envio podem ser consultados na Figura 5.18.

5.6 Máquina de estados e sistema final

Relativamente à máquina de estados, onde o sistema está assente, foi desenhado um enumerador onde estão presentes todos os seus estados possíveis. Na função *main* existe um *switch loop* onde o objeto referente a esse enumerador é constantemente verificado fazendo com que a aplicação averigue constantemente o estado atual.

```
typedef enum STATE {
    IDLE = 0,
    ACTIVE,
    NO_MOVEMENT,
    MOVEMENT_DETECTED
} STATE;
```

Figura 5.20 - Enumerador STATE

Assim, consegue-se atribuir diferentes estados ao objeto do tipo STATE. Poderia ter sido criada uma instância de um objeto do tipo inteiro e com isto atribuir valores fixos aos estado mas esta abordagem facilita a perceção ao programador e de quem analisa o código posteriormente.

```
/* ---------------- STATES ---------------- */
STATE currentState = IDLE;
```

Figura 5.21 - Objeto do tipo STATE

Como está demonstrado na figura em cima, o primeiro estado atribuído é o IDLE e o sistema executará a aplicação com esta predefinição.

Já na figura em baixo é demonstrada a implementação do *switch loop* e posteriormente irá ser exibida cada uma das funções chamadas nos quatro casos. Note que após a inicialização das configurações base relativas às prioridades atribuídas no NVIC pela função init(), da atribuição do IP e porta de destino ao objecto Endpoint, a configuração e habilitação do acelerómetro pela função config_and_start_acc() encontramos, finalmente, a inicialização do timer *keep*_alive relativo ao estado predefinido na primeira execução da aplicação (IDLE).

```
int main()
{
    init();
    wait(1);

    server.set_address("255.255.255.255", 50000);

    config_and_start_acc();

    T_ka.reset();
    T_ka.start();

    while(1) {
        switch ( currentState ) {
            case IDLE:
                idle_state();
                break;
            case ACTIVE:
                active_state();
                break;
            case NO_MOVEMENT:
                no_movement_state();
                break;
            case MOVEMENT_DETECTED:
                movement_detected_state();
                break;
        }
    }
}
```

Figura 5.22 - *Switch loop* presente na função main()

Com este *loop*, presente na função *main*, o processador fica então restringido à comutação dentro destas quatro funções: idle_state(), active_state(), no_movement_state() e movement_detected_state(). Cada uma destas funções implementa o comportamento que o sistema deve cumprir tal como especificado nos requisitos do sistema em 3.1 Requisitos do sistema.

5.6.1 Estado IDLE

No estado IDLE verificamos se o timer, que já está a ser incrementado, contém um valor igual ou superior ao valor predefinido na macro relativa ao tempo de *keep_alive*. Se esta condição se verificar é realizada a reinicialização ao temporizador, ligado o módulo sem-fios, estabelecida a conexão ao ponto de acesso, executado o varrimento e enviados os dados já formatados e finalmente desconectado o módulo. Note que esta ação presente dentro da condição necessita de, no mínimo, aproximadamente onze segundos. Posto isto, os tempos de *keep alive* inferiores a onze segundos serão impossíveis de cumprir. Esta limitação acaba por

ser facilmente transposta visto que estes tempos serão, em princípio, maiores que o valor mínimo da operação com a finalidade de poupar recursos energéticos.

```
void idle_state ( void )
{
    if(T_ka.read() >= IDLE_KEEPALIVE) {
        T_ka.reset();
        T_ka.start();
        configure_and_connect();
        Wifi_GetScanResults("IDLE", false);
        disconnect_and_stop();
    }
}
```

Figura 5.23 - Função relativa ao estado IDLE

5.6.2 Estado MOVEMENT_DETECTED

A implementação relativa a este estado é constituída inicialmente por uma verificação booleana da vaiável movement_detected. Se esta condição se verificar são desabilitadas as interrupções e enviados dois varrimentos, um com o campo TYPE como MOVEMENT_DETECTED e TRIGGER a 1 e o segundo contendo no campo TYPE a *string* ACTIVE, indicando qual o próximo estado que o sistema irá transitar. O enumerador currentState do tipo STATE irá possuir o valor referente ao próximo estado, neste caso ACTIVE, as interrupções voltarão a ser habilitadas e a variável booleana será reposta como *false*. Finalmente são colocados os timers referentes ao próximo estado a operar, sendo incrementados pelo *systick*, após a chamada do método start(). Esta operação reflete-se tanto no *timer* relativo aos varrimentos constantes pelo objeto T_active como também pelo *timer* responsável pelo *timeout* pelo objeto T_timeout.

É de salientar que como a frequência de pesquisas dos pontos de acesso e posterior envio irá aumentar foi então decidido manter a conexão ao ponto de acesso até o estado do sistema voltar a IDLE. Com isto conseguimos que o sistema tenha uma resposta mais rápida quando os valores parametrizados são menores.

```
void movement_detected_state ( void )
{
    if(movement_detected) {

        int_acc.disable_irq();

        configure_and_connect();
        Wifi_GetScanResults("MOVEMENT_DETECTED", true);
        Wifi_GetScanResults("ACTIVE", false);
        currentState = ACTIVE;
        int_acc.enable_irq();
        movement_detected = false;

        T_active.reset();
        T_active.start();

        T_timeout.reset();
        T_timeout.start();
    }
}
```

Figura 5.24 - Função relativa ao estado MOVEMENT_DETECTED

5.6.3 Estado ACTIVE

Na função respeitante ao estado ACTIVE são executadas três verificações a condições. Primeiramente é realizada a averiguação relativa ao estado do temporizador *timeout*, isto é, se o valor do temporizador T_timeout conter um valor maior que o valor especificado, em segundos, na macro TIMEOUT então o objeto currentState irá ser atualizado com o próximo estado a transitar. Tanto o temporizador T_timeout como T_active foram desabilitados visto que não são necessários até ao sistema voltar ao estado ativo.

Já na segunda condição, se a decisão anterior não se verificar, é averiguado se o valor do temporizador T_active é igual ou superior ao valor parametrizado na macro ACTIVE_TRANSMISSION. Nesta condição é habilitado o módulo, estabelecida a conexão com o ponto de acesso predefinido, caso esta tenha sido perdida, e enviado, com a periodicidade definida na macro referida anterior e incluída na compilação do código, um pacote de dados contendo a pesquisa de redes sem-fios nessa área com o campo TYPE contendo o valor "ACTIVE" e TRIGGER a 0. O valor do temporizador T_active é zerado para que, caso o sistema permaneça neste estado até ao próximo envio de dados, poderá então voltar a entrar novamente nesta condição.

Na última condição é verificada a variável global movement_detected. O estado desta variável é alterado para *true* na função que faz *handling* da interrupção responsável pelo movimento detetado sentido pelo acelerómetro. Posto isto, caso tenha sido detetado movimento

por parte do acelerómetro é executado exatamente o mesmo procedimento que na condição explicada anteriormente com a exceção do temporizador T_timeout também ser zerado, o campo TRIGGER conter o valor 1 e a variável movement_detected ser reposta com o valor *false*.

```
void active_state ( void )
{
    if(T_timeout.read() >= TIMEOUT) {

        T_active.stop();
        T_timeout.stop();

        currentState = NO_MOVEMENT;

        return;
    }

    if(T_active.read() >= ACTIVE_TRANSMISSION) {
        configure_and_connect();
        Wifi_GetScanResults("ACTIVE", false);
        T_active.reset();
    }

    if(movement_detected) {
        T_timeout.reset();
        T_active.reset();
        configure_and_connect();
        Wifi_GetScanResults("ACTIVE", true);
        movement_detected = false;
    }
}
```

Figura 5.25 - Função relativa ao estado ACTIVE

5.6.4 Estado NO_MOVEMENT

Muito sucintamente, neste estado são enviados os dados provenientes dos varrimentos das redes sem fios num determinado número de vezes definido pelo utilizador. Este número de vezes é definido pelo parâmetro N_MOVEMENTS. Inicialmente é realizada a habilitação do modulo e a associação ao ponto de acesso predefinido, caso a ligação se tenha perdido, após isto é efetuado um *for loop* contendo, em cada execução, um varrimento e posterior envio de dados com o campo TYPE incluindo o valor “MOVEMENT_DETECTED”. Este *loop* poderá ser interrompido caso a variável booleana global possua o valor *true* a qualquer momento. Após esse *loop* ter sido executado é novamente enviado outro varrimento diferenciando apenas o facto do campo TYPE possuir a *string* referente ao próximo estado: “IDLE”. Finalmente, é

habilitado o timer *keep alive* (T_ka), a desassociação ao ponto de acesso e, posteriormente, o término do funcionamento do módulo CC3000. O objeto currentState irá ser atualizado para o valor IDLE.

```
void no_movement_state ( void )
{
    configure_and_connect();
    for(int i=0 ; i < N_MOVEMENTS ; i++) {
        if(movement_detected) {
            currentState = MOVEMENT_DETECTED;
            return;
        }

        else Wifi_GetScanResults("NO_MOVEMENT", false);
    }

    Wifi_GetScanResults("IDLE", false);

    T_ka.reset();
    T_ka.start();
    disconnect_and_stop();

    currentState = IDLE;
}
```

Figura 5.26 - Função relativa ao estado NO_MOVEMENT

5.6.5 Handler da interrupção

Apesar de esta função não estar diretamente envolvida na máquina de estados presente no *switch loop*, como demonstrado na Figura 5.22, esta tem um papel deveras importante quando nos referimos às transições de estados.

Nesta função de *handling* conseguimos com que seja efetuada uma mudança de contexto no ponto de vista do microcontrolador quando é despoletado um *trigger* e acionada a interrupção. Posto isto, nesta interrupção o temporizador relativo ao *keep alive* é desabilitado e a variável global movement_detected obterá o valor *true*. Caso o estado atual seja o IDLE o próximo estado será o MOVEMENT DETECTED.

```
void funcao()
{
    T_ka.stop();
    if(currentState == IDLE ) {
        currentState = MOVEMENT_DETECTED;
    }
    movement_detected = true;
}
```

Figura 5.27 - Função *callback* acelerómetro

Capítulo 6 Resultados

Após as fases de análise, *design* e implementação é necessário apresentar todos os resultados relevantes relativos ao sistema desenvolvido. Os resultados serão divididos em subtópicos e explicados e comentados svieparadamente, denotando assim a importância de casa um destes no sistema final.

6.1 Pesquisa de conexões sem-fios (*Scan*)

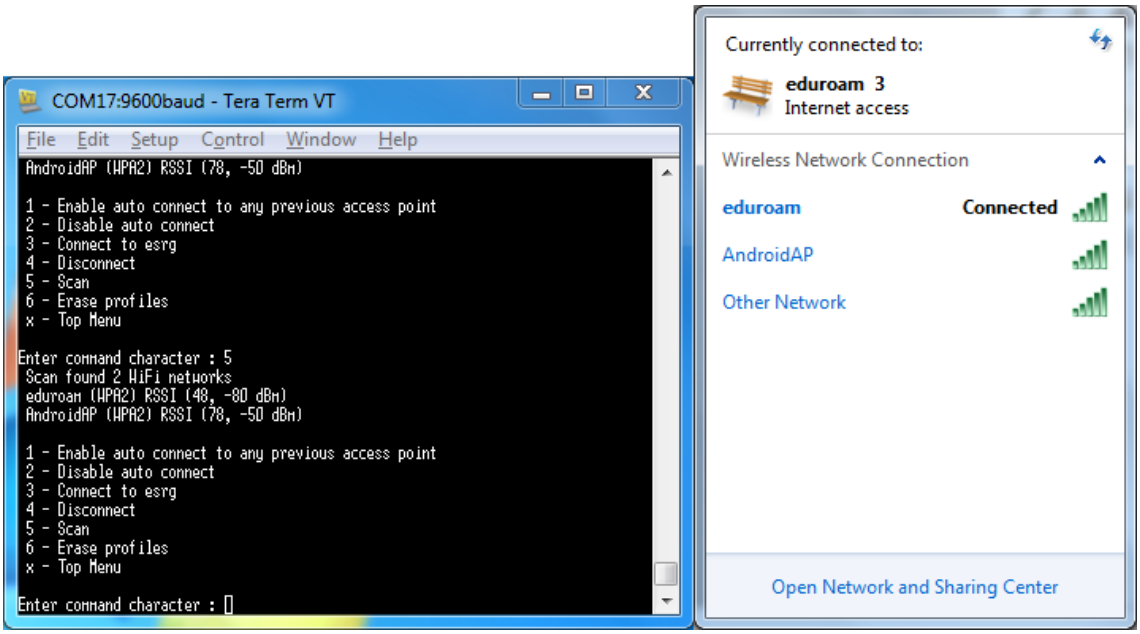


Figura 6.1 - *Scan* das redes vizinhas. CC3000 vs Broadcom 802.11g Adapter

Este resultado é adquirido a partir das redes disponíveis na vizinhança da *tag* ligada à COM17 e com a *driver* USB-CDC instalada. As redes 802.11 detetadas pelo sistema operativo Windows 7, à direita, correspondem às redes obtidas e mostradas no terminal, à esquerda. Como podemos notar, existe uma certa disparidade relativa à força de sinal das redes 802.11 adquiridas nos dois dispositivos, sendo que este resultado prende-se ao facto do *WLAN card* do HP dv6 1250-sp ter uma maior sensibilidade.

6.2 Modos de leitura por *pooling* e Interrupção do Acelerómetro

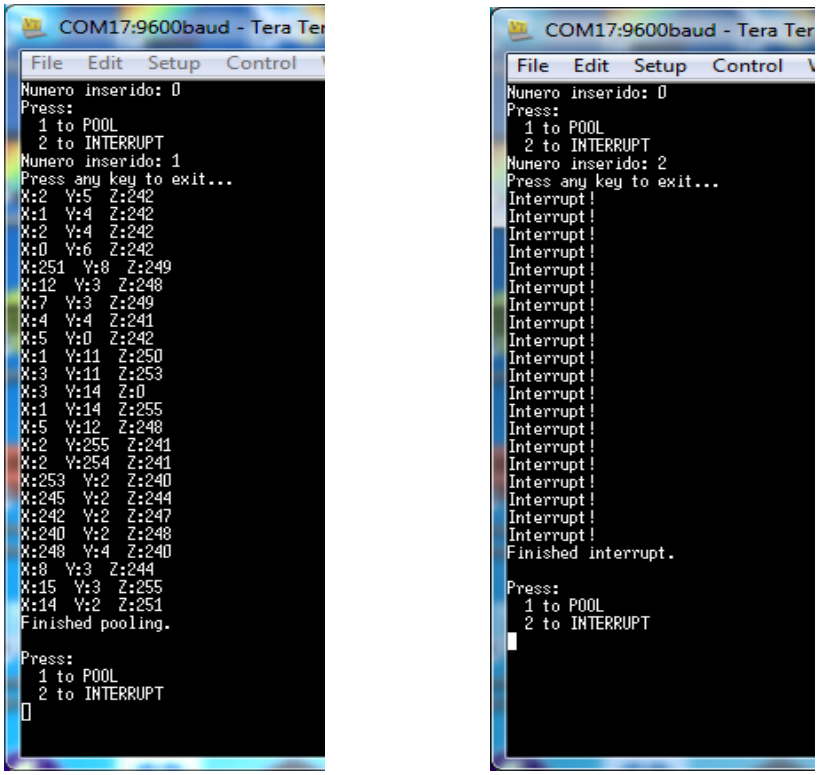


Figura 6.2 - Modos de operação do acelerómetro

Relativamente ao acelerómetro são mostrados os valores adquiridos através do método de leitura dos registos axiais por pooling, à esquerda, e interrupção por deteção de movimento num determinado valor de *threshold*, à esquerda.

6.3 Envio em *broadcast* de pacotes UDP

É mostrada, na Figura 6.3, a implementação do envio de pacotes UDP em *broadcast*. Assim, foram coletadas três janelas em que o terminal presente no canto superior esquerdo representa o *interface* com a *tag* e CC3000, canto superior direito é um servidor UDP a correr na máquina "Joao-PC" (OS Windows 7 com emulador Cygwin) na porta de chegada 50000 e finalmente a janela inferior corresponde a um software *free and open-source packet analyser* denominado por Wireshark. Verifica-se também que foram enviados dois pacotes no terminal relativo à *tag* e foram recebidos pelo servidor UDP e também pelo Wireshark, ambos executados no PC, os dois pacotes UDP referentes ao envio.

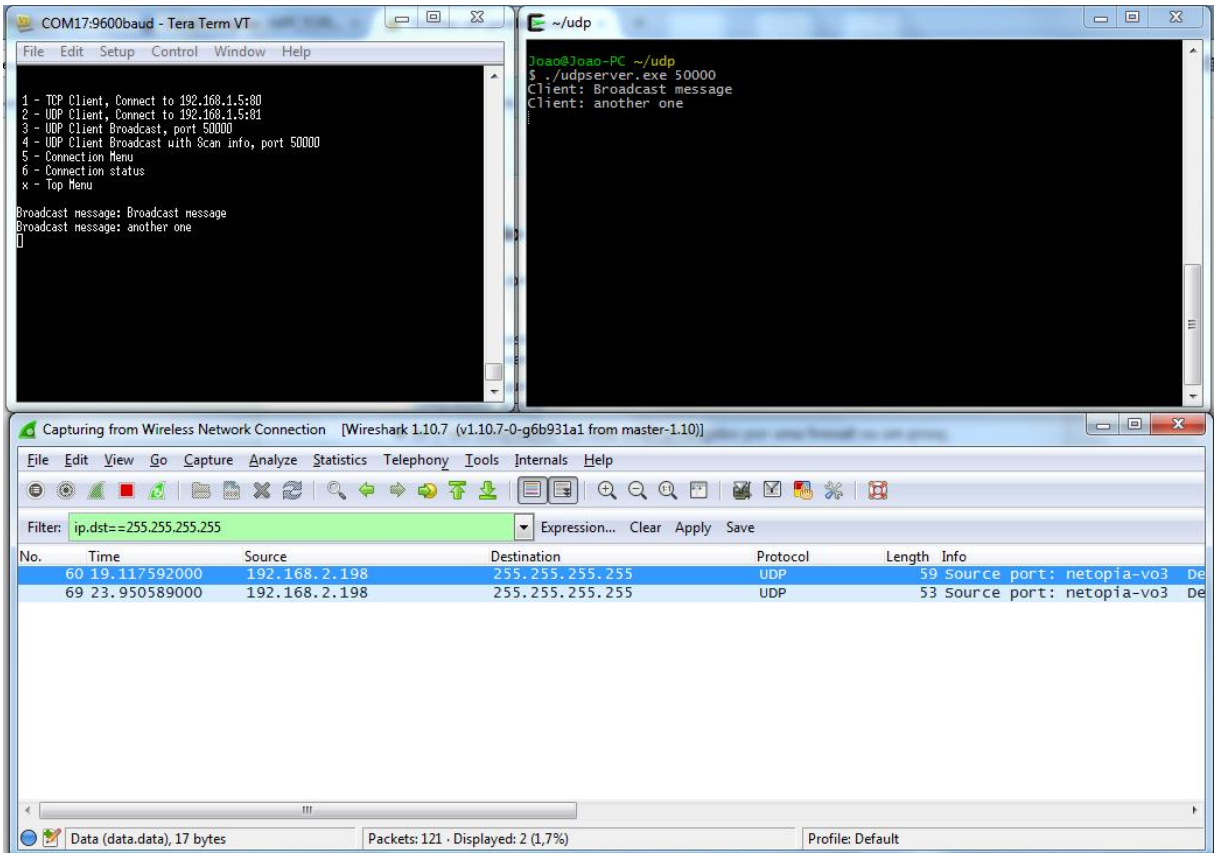


Figura 6.3 - Envio UDP em *broadcast*

6.4 Consumos WiFi DipCortex (ARM Cortex M3 & CC3000)

Com os vários blocos testados e com resultados bastante positivos foi decidido efetuar medições ao sistema em variados casos. Todas estas medições foram realizadas com um multímetro digital Keithley 2100 Digital Multimeter 6-1/2 disponibilizado pelo ESRG, tendo este uma precisão de seis dígitos. Estes valores foram retirados por *software* e carregados para uma folha Windows Office Excel com ajuda de uma *driver* e *add-on* Excel. Todas as figuras seguintes apresentam valores de corrente DC medidos nos terminais da placa de desenvolvimento WiFi DipCortex em que foram retiradas aproximadamente 15 leituras por segundo num total de 600 leituras.

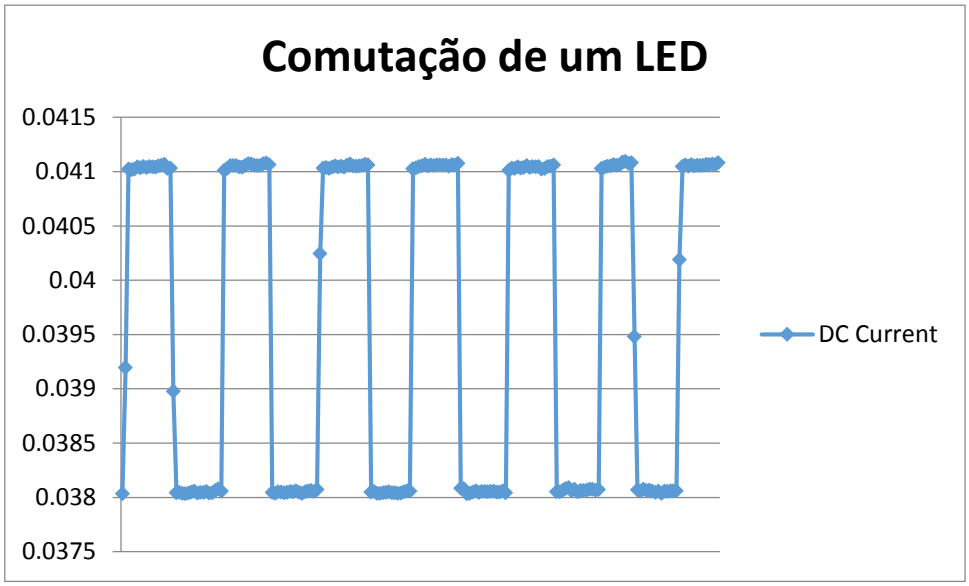


Figura 6.4 – Consumo relativo à comutação de um *port* (PB20) em que neste está ligado um LED comercial e uma resistência em série

Neste primeiro gráfico pode-se visualizar o consumo de todo o sistema quando um LED, ligado a um porto do microcontrolador, é comutado. Este resultado mostra que o consumo do LED é expresso pela equação em baixo. ($R_{pull\text{-}up}$ = 1000 Ω)

$$\Delta I_{led} = I_{sistema\ e\ led} - I_{sistema} = \sim 3mA$$

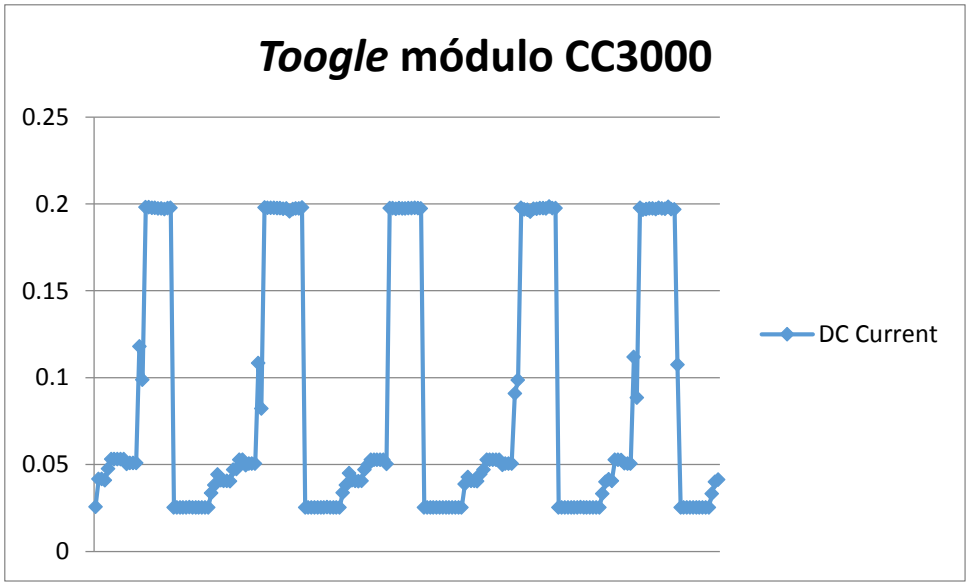


Figura 6.5 - Consumo comutando a alimentação elétrica do módulo *wireless* CC3000

Na Figura 6.5 concluímos que o módulo CC3000, quando ligado, aumenta o consumo energético do sistema para 400%. Isto deve-se aos requisitos necessários e especificidades do próprio *hardware*.

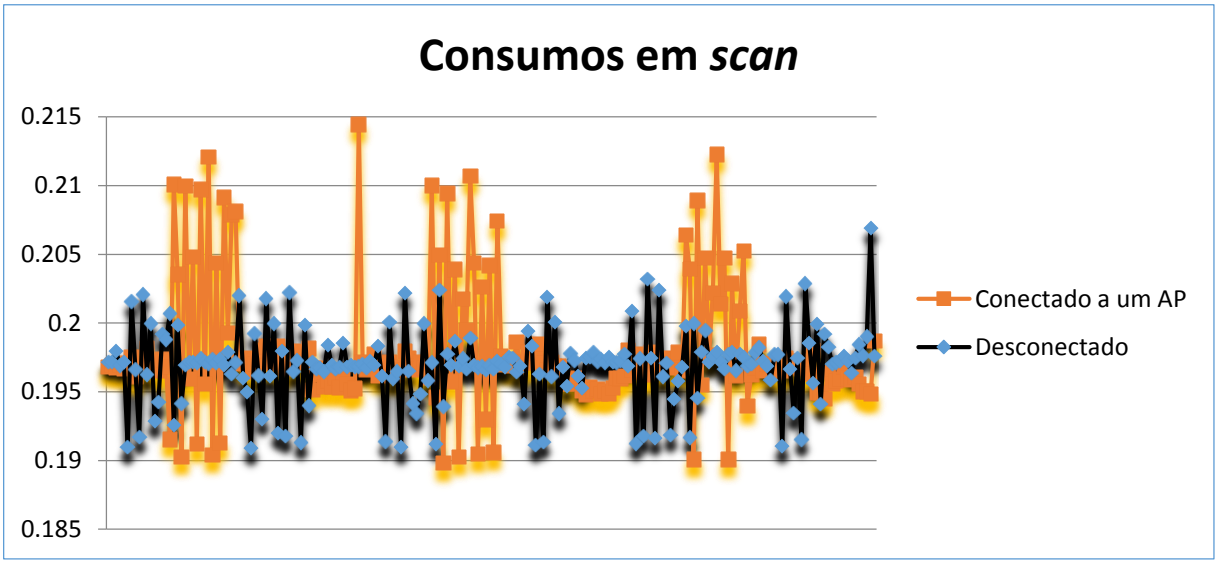


Figura 6.6 - Diferença entre consumos de varrimentos quando o processador CC3000 está conectado ou desconectado a um AP

Com o gráfico ilustrado pela Figura 6.6 concluímos que existe uma grande diferença quando se realizam varrimentos às redes enquanto o dispositivo está conectado ou desconectado a um ponto de acesso. Esta diferença não só se verifica em termos do diminuído número de vezes que o este processo é realizado durante o mesmo tempo como também pelos picos de consumos energéticos mais elevados quando o processo é realizado com conexão estabelecida comparativamente ao processo realizado com ausência de conexão. Em valores, a potencialidade de frequência de varrimentos efetuados quando a conexão está estabelecida desce para 50% (6 varrimentos sinalizados em cor negra, 3 varrimentos a cor-de-laranja) e a os picos de corrente aumentam $\sim 8{,}5\ mA$.

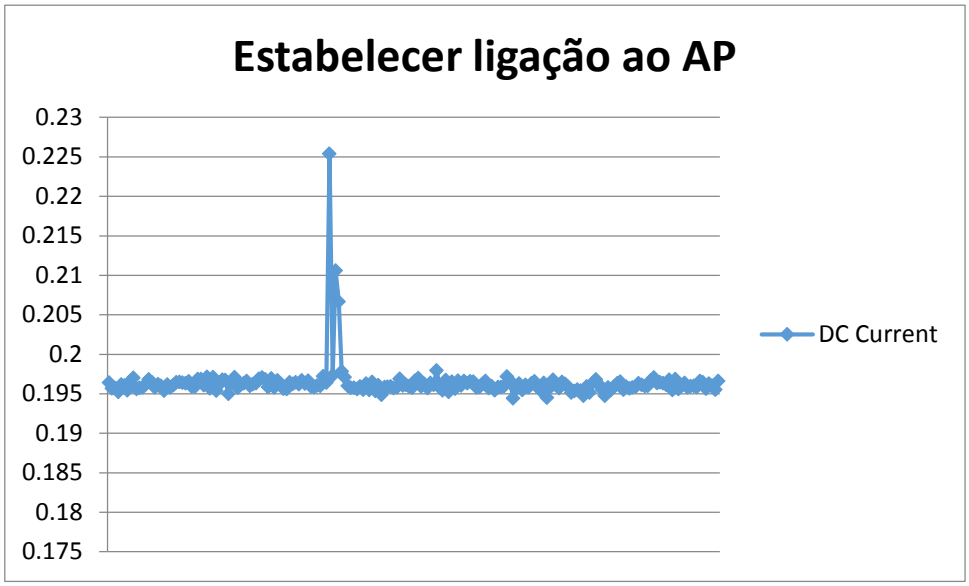


Figura 6.7 - Pico de consumo energético verificado quando a ligação a um AP está a ser estabelecida

No que toca ao estabelecimento da ligação a um ponto de acesso IEEE 802.11 predefinido, captou-se um pico de corrente ao realizar o chamado *handshake* para estabelecer a ligação. Neste gráfico verifica-se um rápido pico $\sim 255\ mA$.

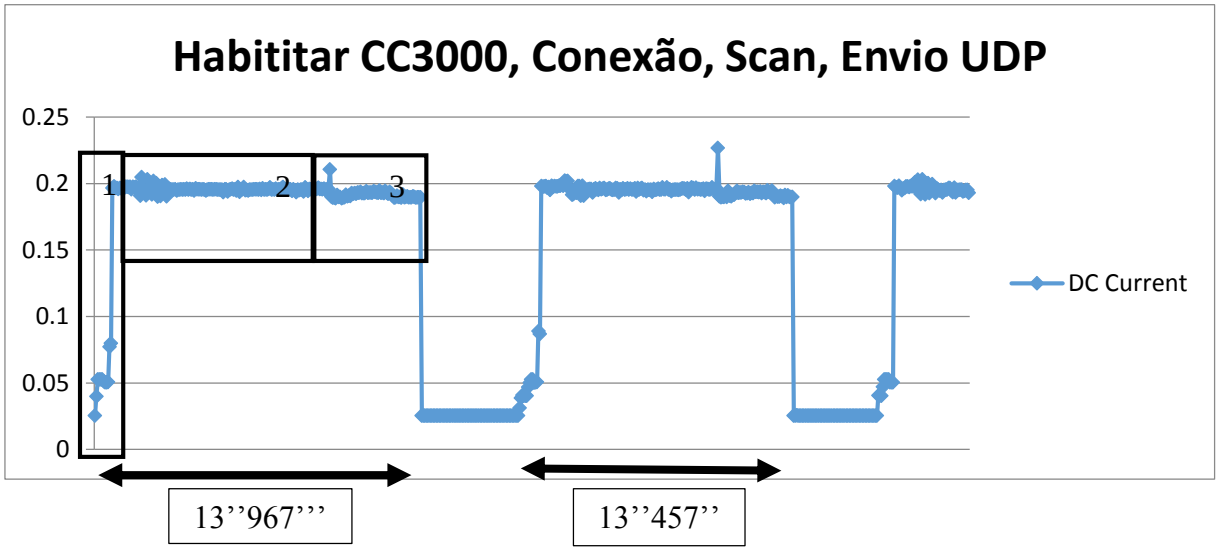


Figura 6.8 - Consumo energético quando é executada a habilitação do processador *wireless*, estabelecida a ligação a um AP, varrimentos de redes 802.11, envio do pacote em *broadcast* e, finalmente, desabilitação do CC3000

Neste último gráfico é mostrado o consumo da placa WiFi DipCortex ao efetuar todas as ações necessárias para enviar um pacote UDP em *broadcast*. Na zona número 1 verificamos a habilitação do módulo Wi-Fi Texas CC3000, na zona número 2 verificamos o comportamento do sistema ao executar a pesquisa de pontos de acesso e, finalmente, na zona número 3 está contido o envio do pacote UDP.

6.4.1 Estimativa da atividade do sistema quando opera alimentado a bateria

Quando o sistema permanece no estado IDLE são enviados pacotes de informação para o servidor com uma certa periodicidade. Assim, medindo os consumos energéticos desta atividade periódica podemos prever qual é o tempo máximo que o nosso sistema pode permanecer ativo estando este alimentado eletricamente por uma bateria. Posto isto e com ajuda dos resultados obtidos anteriormente, temos a seguinte imagem abaixo que relaciona o consumo energético expresso em A (amperes) por tempo t (segundos).

Sabendo que cada demora aproximadamente 13.5 segundos em que, durante esse tempo, o seu consumo ronda os 0.196 A e em *stand-by* ronda os 0.0255 A, conseguimos assim perspetivar um consumo médio dependendo do tempo IDLE escolhido. Suponhamos então que o tempo em IDLE escolhido é de 30 segundos. O consumo médio pedido à bateria é dado por:

$$Consumo_{médio} = \frac{I_{IDLE} * t_{IDLE} + I_{envio} * t_{envio}}{T}$$

Em que I_{IDLE} neste exemplo o consumo médio do sistema em IDLE (0.0255 A), t_{IDLE} o tempo escolhido para o temporizar T_KA (30 s), I_{envio} representa o consumo médio do sistema enquanto executa o envio (0.196 A), t_{envio} apresenta o tempo mínimo necessário para enviar um pacote de dados (13.712 s) e, finalmente, a variável T é o período da atividade (30 s+13.712 s).

$$Consumo_{médio} = \frac{0.0255 * 30 + 0.196 * 13.712}{30 + 13.712} = 78.984\ mA$$

Supondo que a bateria utilizada tem uma capacidade de 1000 mAh, conseguimos prever que a duração aproximada do sistema em pleno funcionamento é de:

$$Duração\ aproximada = \frac{1000}{78.984} = 12.66\ horas$$

Alcançamos então, com um tempo T_KA de 30 segundos, o sistema em funcionamento durante ~ 12 horas e 40 segundos.

Esta estimativa teórica parte da escolha de um tempo IDLE de 30 segundos e de uma bateria com capacidade de 1000 mAh. Outro ponto essencial é o facto de quantas mais vezes for atuado o sistema, de maneira a entrar no estado ativo, maior será o seu consumo médio,

fazendo com que a precisão desta estimativa seja inversamente proporcional ao número de deteções de movimentos no sistema.

Para aumentar a longevidade da bateria, quando o sistema é eletricamente alimentado por esta, podemos então proceder à escolha de um tempo T_KA maior ou escolher uma bateria de capacidade mais elevada. Existem outros parâmetros que podem ser considerados como por exemplo, um tempo em que o acelerómetro é desabilitado logo depois de transitar do estado NO_MOVEMENT para o estado IDLE, fazendo com que o sistema não volte logo a entrar no estado MOVEMENT_DETECTED e permaneça um pouco mais no estado IDLE, reduzindo o consumo do sistema.

6.5 Teste em funcionamento perdurado

Foi realizado um teste à operacionalidade do sistema, ligado a uma fonte de energia ininterrupta, de maneira a validar o seu funcionamento durante um certo período de tempo, comprovando que este permanece ativo e funcional durante esse intervalo predefinido. Para isso foi decidido colocar este sistema em funcionamento durante vinte e quatro horas. Na figura em baixo pode ser comprovado este resultado.

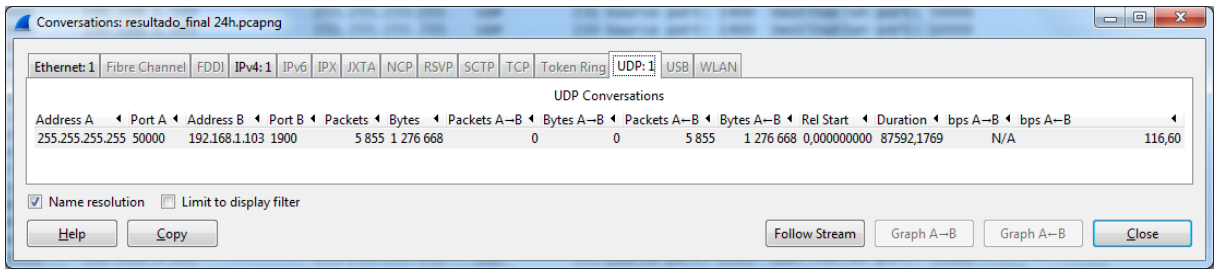


Figura 6.9 - Resultado final do tete - contagem de pacotes recebidos e duração

Com esta imagem conseguimos demonstrar que o protocolo utilizado para enviar os pacotes foi o UDP e que o método de envio foi o *broadcast* (Endereço de destino: 255.255.255.255) para a porta de destino 50000. A duração de todo este processo foi de 87592.1769 segundos equivalente a 24 horas e 19 minutos.

$$t_{stress\ testing\ horas} = \frac{t_{stress\ testing\ segundos}}{3600} = \frac{87592}{3600} = \sim 24{,}3\ horas$$

Foi assim provado que o sistema opera sem interrupções e em pleno funcionamento durante 24 horas, onde foram enviados para o servidor 5855 pacotes e 1276668 bytes, fazendo com que, em média, se tenham enviado ~ 218 bytes/pacote. Na figura abaixo pode-se consultar todo o conteúdo do último pacote recebido em formato binário e também em ASCII. Esse conteúdo contém protocolo JSON inserido num UDP *datagram payload*.

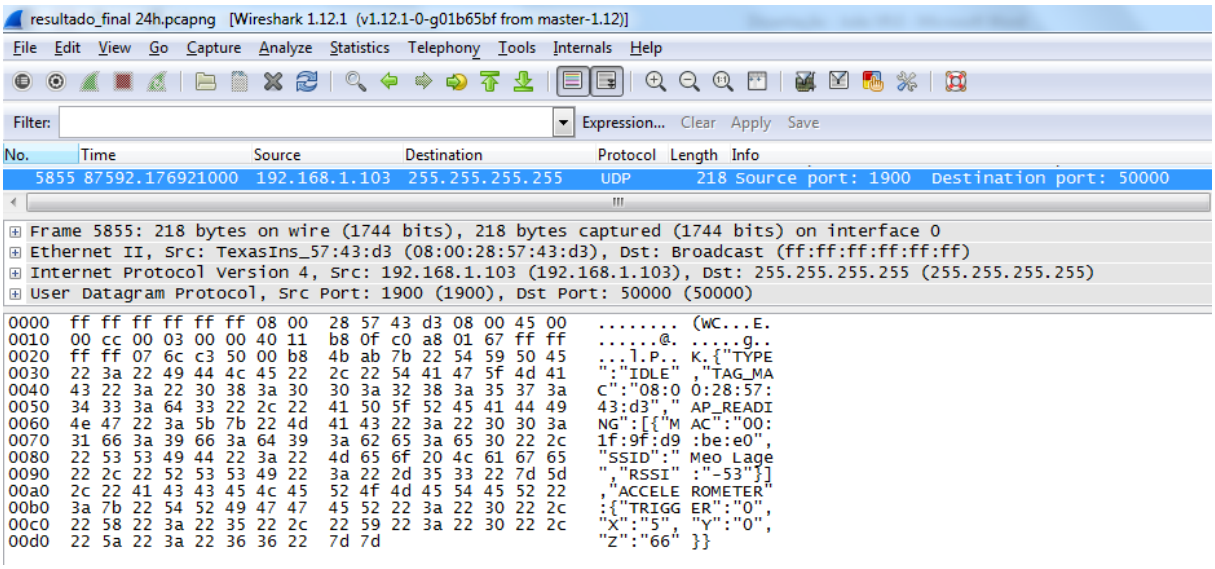


Figura 6.10 - Ultimo pacote recebido e seu conteúdo

Capítulo 7 Conclusão

Ao longo desta dissertação de mestrado foi analisado um possível sistema a desenvolver, desenhadas as especificidades, estudados os variados componentes teóricos que teriam que ser integrados no trabalho e, finalmente, implementado o protótipo final.

Este projeto, contendo vários campos de conhecimento distintos, ampliou o meu conhecimento em diversas formas e todo o apoio necessário foi prestado de forma procurar a melhor solução possível para solucionar os problemas encontrados. É também necessário relevar a desafiante e importante experiência adquirida com a entreajuda e o espirito de equipa onde estes foram altamente aprimorados no decorrer do projeto. Este plano multidisciplinar foi elevadamente educativo no que toca à pesquisa e implementação de uma solução para recolha de dados para posterior monitorização e localização *indoor* que hoje em dia está muito em voga.

Para além dos conhecimentos e habilidades adquiridas, todos os objetivos foram devidamente conseguidos em tempo útil, sendo que as decisões tomadas ao longo da implementação efetuaram sucesso no projeto e no protótipo.

Vídeo do protótipo funcional aqui.

7.1 Limitações

Em termos das limitações obtidas estas são muito reduzidas. Para solucionar parte dos obstáculos encontrados, foi atualizado o *firmware* do módulo Texas Instruments CC3000 o que faz com que certos *bugs* tenham sido corrigidos. Este *firmware* pode ser obtido no sítio *online* da SolderSpash como também todas as instruções necessárias.

Outra limitação foi o facto do *debug* a este tipo de produtos mbed não estar muito bem desenvolvido e documentado. Apesar de todas as ferramentas fornecidas, tais como o IDE, ferramentas de revisão e programação do microcontrolador, existe uma grande lacuna em termos de *debug* onde este terá que ser feito com uma ferramenta externa e para a qual ainda não existe muita informação. Assim, alguns dos métodos utilizados para realizar esta operação foi através de um LED ou então por comandos série (USART), dos quais não são os melhores para leitura de registos internos, apontadores, *breakpoints* e saltos entre funções.

7.2 Trabalho futuro

Como trabalho futuro existem várias ideias que este protótipo poderá incluir de forma a permitir melhoramentos significativos. O primeiro grande aperfeiçoamento seria a inclusão de uma configuração sem-fios de modo a escolher qual o ponto de acesso, servidor e porta para onde enviar os dados das leituras. Assim esta configuração poderia ser realizada sem ter que alterar o código fonte e posterior necessidade de reprogramação do microcontrolador.

Outra ideia prende-se com a necessidade de um circuito responsável pela gestão de bateria e carregamento da mesma por ligação universal USB. Assim, o sistema conseguiria enviar informações relativas à necessidade de carregamento da bateria caso esta esteja com pouca energia. Com este circuito e o utilizador sendo avisado, poderia então conduzir o módulo a um posto de carregamento e proceder ao carregamento para que o sistema mantenha o seu funcionamento habitual sem interrupções.

Por último, para que se consiga integrar este sistema num contexto de maior dimensão, dever-se-ia estudar a possibilidade de conectar este tipo de dispositivo aos pontos de acesso "eduroam" da Universidade do Minho que necessitam de um método de encriptação AES e um método de autenticação PEAP. Uma rede com estas especificidades necessita de compatibilidade por parte do cliente.

REFERÊNCIAS

[1] N. R. Storey, *Safety Critical Computer Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1996.

[2] J. M. C. E. da Lage, "RODOS gateway frame over ZigBee communication protocol," Julius-Maximilian University of Würzburg – GERMANY, 2014.

[3] H. Wang, H. Zhou, and L. Zhu, "Analysis and research on indoor positioning method based on IEEE 802.11," *... , Netw. Mob. ...*, pp. 2181–2183, 2007.

[4] S.-T. Sheu, M.-S. Li, Y.-H. Tsai, and P.-M. Lin, "In-door wireless positioning technology," *Proc. 5th Int. ICST Conf. Commun. Netw. China*, 2010.

[5] P. Brida, J. Benikovsky, and J. Machaj, "Performance investigation of WifiLOC positioning system," *2011 34th Int. Conf. Telecommun. Signal Process.*, pp. 203–207, Aug. 2011.

[6] T. King, T. Butter, M. Brantner, S. Kopf, T. Haenselmann, A. Biskop, F. Andreas, and W. Effelsberg, "Distribution of Fingerprints for 802 . 11-based Positioning Systems," pp. 224–226, 2007.

[7] R. Hansen and R. Wind, "Algorithmic strategies for adapting to environmental changes in 802.11 location fingerprinting," *Indoor Position. ...*, no. September, pp. 15–17, 2010.

[8] K. Kaemarungsi, "Distribution of WLAN received signal strength indication for indoor location determination," *Wirel. Pervasive Comput. 2006 1st ...*, 2006.

[9] P. Bahl and V. N. Padmanabhan, "RADAR: an in-building RF-based user location and tracking system," *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2. pp. 775–784 vol.2, 2000.

[10] H. Wang, L. Ma, Y. Xu, and Z. Deng, "Dynamic Radio Map Construction for WLAN Indoor Location," *2011 Third Int. Conf. Intell. Human-Machine Syst. Cybern.*, pp. 162–165, Aug. 2011.

[11] H. Miao, Z. Wang, and J. Wang, "A novel access point selection strategy for indoor location with Wi-Fi," *Control Decis. ...*, no. 61174059, pp. 5260–5265, 2014.

[12] Wikipedia, "Euclidean distance." [Online]. Available: http://en.wikipedia.org/wiki/Euclidean_distance. [Accessed: 15-Oct-2014].

[13] Wikipedia, "k-nearest neighbors algorithm." [Online]. Available: http://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm. [Accessed: 15-Oct-2014].

[14] M. M. Atia and M. Korenberg, “A Consistent Zero-Configuration GPS-Lil (e Indoor Positioning System Based on Signal Strength in IEEE,” pp. 1068–1073, 2012.

[15] Cisco, “Connecting Switches and Ethernet Technology.” [Online]. Available: http://www.ciscopress.com/articles/article.asp?p=1276662&seqNum=3. [Accessed: 05-Nov-2014].

[16] Cisco, “Ethernet Technologies.” [Online]. Available: http://docwiki.cisco.com/wiki/Ethernet_Technologies. [Accessed: 05-Nov-2014].

[17] Wikipedia, “Data link layer.” [Online]. Available: http://en.wikipedia.org/wiki/Data_link_layer. [Accessed: 05-Nov-2014].

[18] Wikipedia, “Media access control.” [Online]. Available: http://en.wikipedia.org/wiki/Media_access_control. [Accessed: 05-Nov-2014].

[19] Wikipedia, “Logical link control.” [Online]. Available: http://en.wikipedia.org/wiki/Logical_link_control. [Accessed: 05-Nov-2014].

[20] Wikipedia, “Wi-Fi.” [Online]. Available: http://en.wikipedia.org/wiki/Wi-Fi. [Accessed: 05-Nov-2014].

[21] Wikipedia, “IPv6.” [Online]. Available: http://en.wikipedia.org/wiki/IPv6. [Accessed: 05-Nov-2014].

[22] Wikipedia, “Transport layer.” [Online]. Available: http://en.wikipedia.org/wiki/Transport_layer. [Accessed: 05-Nov-2014].

[23] Wikipedia, “Transmission Control Protocol.” [Online]. Available: http://en.wikipedia.org/wiki/Transmission_Control_Protocol. [Accessed: 05-Nov-2014].

[24] Wikipedia, “User Datagram Protocol.” [Online]. Available: http://en.wikipedia.org/wiki/User_Datagram_Protocol. [Accessed: 05-Nov-2014].

[25] H. M. Nikhilesh Jasuja, Pooja Sehgal, Leonel Guillermo Mendoza Estrada, “TCP vs UDP. Diffen.com.” [Online]. Available: http://www.diffen.com/difference/TCP_vs_UDP. [Accessed: 05-Nov-2014].

[26] ARM mbed, “Handbook | mbed.” [Online]. Available: http://developer.mbed.org/handbook/Homepage. [Accessed: 04-Nov-2014].

[27] NXP, “LPC1315/16/17/45/46/47 32-bit ARM Cortex-M3 microcontroller,” 2012.

[28] Texas Instriments, “CC3000 Host Programming Guide.” [Online]. Available: http://processors.wiki.ti.com/index.php/CC3000_Host_Programming_Guide. [Accessed: 05-Nov-2014].

[29] Texas Instruments, “TI SimpleLink™ CC3000 Module – Wi-Fi 802.11b/g Network Processor,” 2012.

[30] "SolderSplash Labs," "WiFi DipCortex – ARM Cortex Development Board." [Online]. Available: http://www.soldersplash.co.uk/products/wifi-dipcortex/. [Accessed: 05-Nov-2014].

[31] STMicroelectronics, "ultra low-power high performance 3-axes 'nano' accelerometer," 2009.

[32] git, "Getting Started - Git Basics." [Online]. Available: http://git-scm.com/book/en/v2/Getting-Started-Git-Basics. [Accessed: 05-Nov-2014].

[33] V. Baroncini and E. Parente, "WIRELESS – A Norma 802.11b e WAP," *Universidade Federal do Paraná Departamento de Engenharia Elétrica Mestrado em Telecomunicações*, 2001. [Online]. Available: http://www.cricte2004.eletrica.ufpr.br/edu/anterior/cd01/trab/wireless/Wireless.htm.

[34] "802.11," *Indian Institute of Technology, Bombay*. [Online]. Available: http://www.cse.iitb.ac.in/~puru/courses/autumn08/classes/80211.pdf. [Accessed: 05-Nov-2014].

[35] M. Nascimento, "Principais mudanças do IPv6 em relação ao IPv4 para o CCNA," 2011. [Online]. Available: http://www.dltec.com.br/blog/cisco/principais-mudancas-do-ipv6-em-relacao-ao-ipv4-para-o-ccna/. [Accessed: 05-Nov-2014].

[36] P. Arnould and J.-J. Bascou, "Notion de base sur les réseaux - Cisco Networking Academy."

[37] Clker, "Walking person silhouette clip art." [Online]. Available: http://www.clker.com/clipart-14727.html. [Accessed: 05-Nov-2014].

[38] git, "Distributed Git - Distributed Workflows." [Online]. Available: http://git-scm.com/book/en/v1/Distributed-Git-Distributed-Workflows. [Accessed: 05-Nov-2014].

ANEXOS

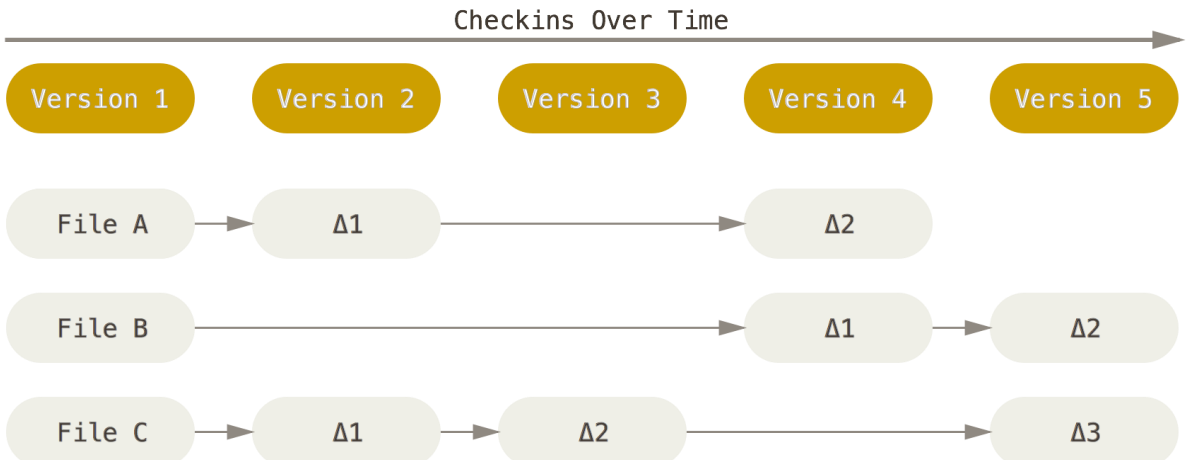
Sistema de controlo de versões - Git

O git é uma rápida e moderna implementação de um sistema de controlo de versões distribuído. (DVCS)

Um sistema de controlo de versões (VCS) permite ao utilizador localizar e seguir o histórico de um conjunto variado de ficheiros armazenando *snapshots* capturados de cada um dos ficheiros criados ou modificados. Para trabalhar em equipa no desenvolvimento de um projeto em equipa é necessário que o repositório Git esteja hospedado num servidor local, em vez de este projeto se localizar em cada uma das máquinas dos colaboradores. O Git é um DVCS no qual todos os seus utilizadores copiam todo o conteúdo do repositório hospedado num servidor local e cada um deles executa a sua implementação independentemente dos outros, promovendo assim o paralelismo de tarefas e reduzindo problemas de conexão. Este sistema fornece também a possibilidade de um utilizador restaurar o servidor em caso de corrupção do repositório armazenado no servidor.

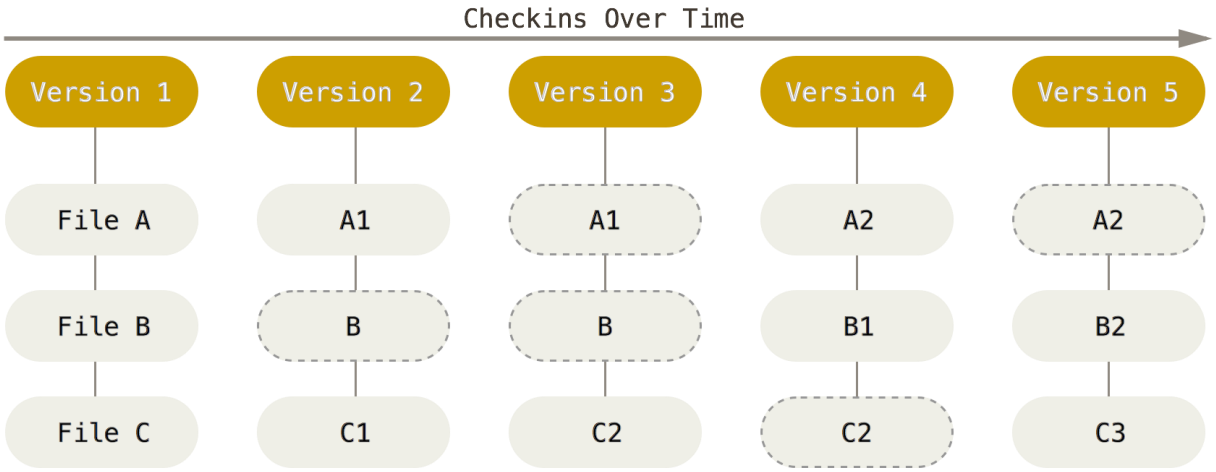
Git vs outros VCS

A maioria de outros VCS (CVS, Subversion, Perforce, Bazaar, entre outros) armazenam informação como uma lista baseada em alterações dos ficheiros no repositório. Esta lista baseada nas alterações cometidas resulta num conjunto de ficheiros que arrecada as alterações em cada ficheiro em função do tempo.



Anexo 1 - Implementação de outros VCS [32]

Já o Git não pensa num sistema de armazenamento deste género. Por outro lado, o Git implementa os dados que armazena como um conjunto de *snapshots* de sistemas de ficheiros em miniatura. Todas as vezes que um utilizador comete as suas utilizações, salva o estado do seu projeto no Git, este sistema apenas tira uma fotografia ao aspeto em que todos os ficheiros se encontram, guardando assim uma referência a esse *snapshot*. Primando pela eficiência, se um projeto cometido tiver ficheiros que não foram modificados, então o Git não os armazenará novamente, apenas irá criar uma ligação (*link*) com o seu antecessor semelhante. O Git não trabalha com ficheiros mas sim com séries de *snapshots*.



Anexo 2 - Método *snapshots* presente no Git [32]

Outra função deveras importante no Git é também a verificação de integridade a partir de *checksum* dos ficheiros, fazendo com que seja impossível alterar os ficheiros sem que o Git não saiba. A integridade é um dos princípios base deste VCS e faz com não se perca informação na transmissão ou os ficheiros recebidos venham corrompidos sem que o Git não detete.

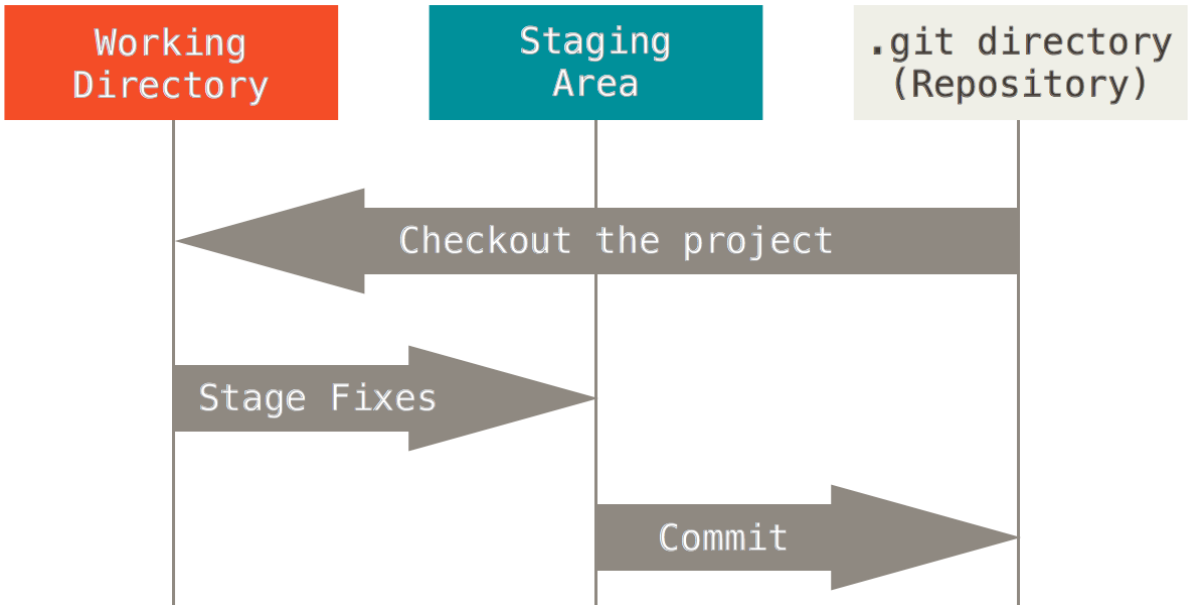
O mecanismo que o Git usa para realizar a soma de verificação tem como nome do algoritmo SHA-1 *hash*. Este algoritmo gera uma chave de quarenta caracteres hexadecimais (0–9 e a–f) e é calculada baseada no conteúdo do ficheiro ou diretoria. [32]

Os três estados Git

O Git é caracterizado pelos três estados mostrados no Anexo 3 - Estados gerais Git **[32]**. O *workflow* deste sistema é caracterizado por:

1. Adiciona-se ficheiros no diretório de trabalho
2. Editam-se os ficheiros, sendo criados *snapshots* destes na *staging área*

3. É realizado um *commit*, e são então armazenados permanentemente os *snapshots* dos ficheiros da *staging area* no repositório Git.



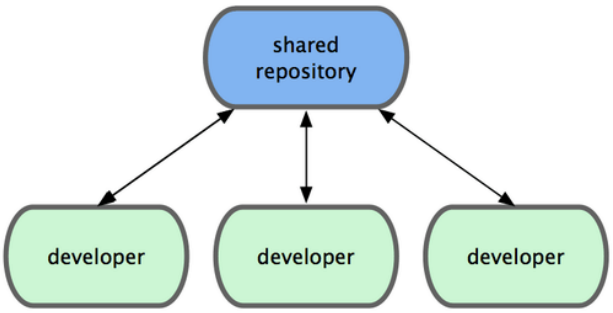
Anexo 3 - Estados gerais Git [32]

Distributed GIT

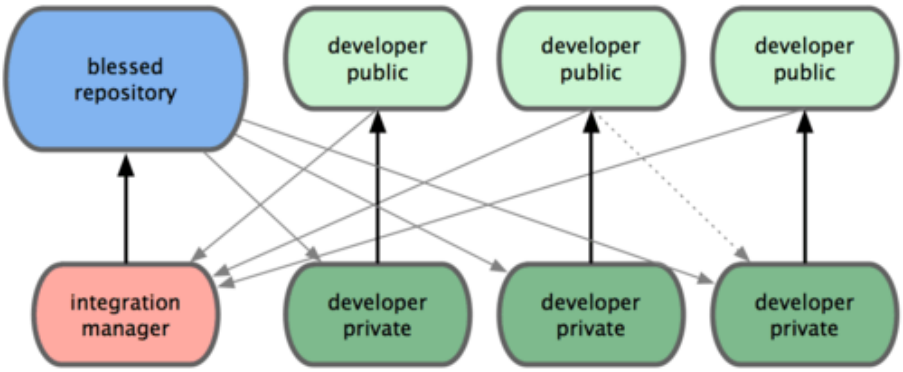
Este sistema faz com que seja possível a sua integração em contextos de desenvolvimento distribuído. Em sistemas centralizados, cada programador é um nó de trabalho, onde realiza trabalho na mesma proporção no *central hub*, já no Git cada utilizador é ao mesmo tempo um nó como também um *hub* – fazendo com que cada programador não só consiga contribuir com o seu trabalho sendo carregado para outros repositórios como também manter o seu repositório público em que os outros utilizadores se consigam basear o seu trabalho e assim contribuir igualmente.

Este ponto de vista abre um vasto leque de possibilidades de trabalho onde um projeto em equipa pode operar. Seguidamente irão ser mostrados alguns paradigmas comuns que operam baseado no Git e visam flexibilidade, organização e hierarquia a um projeto em equipa.

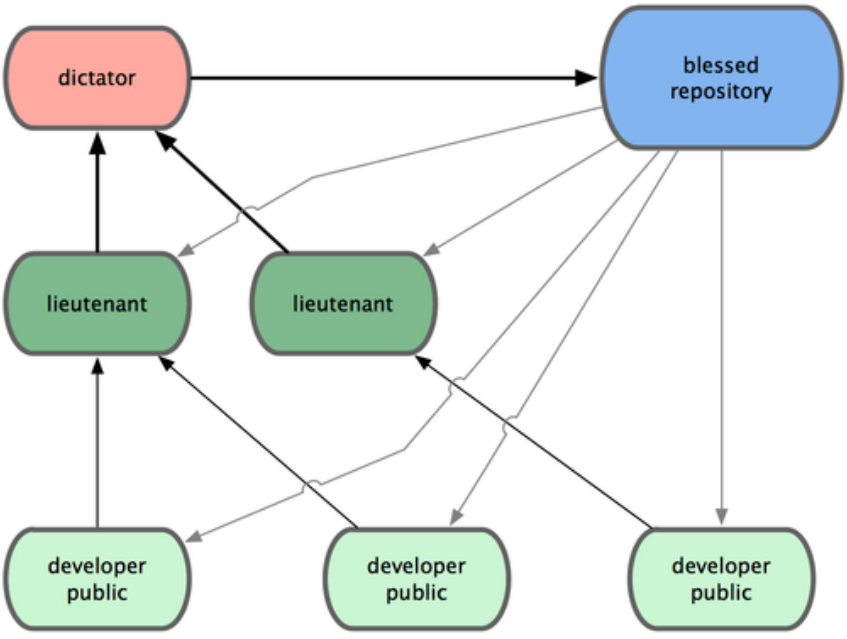
Estes conceitos são baseados em paradigmas muito próprios de sistemas de subversões. Estes modelos trabalham perfeitamente no Git.



Anexo 4 - ***Workflow centralizado*** [38]



Anexo 5 - ***Integration-Manager Workflow*** [38]



Anexo 6 - ***Dictator and Lieutenants Workflow*** [38]

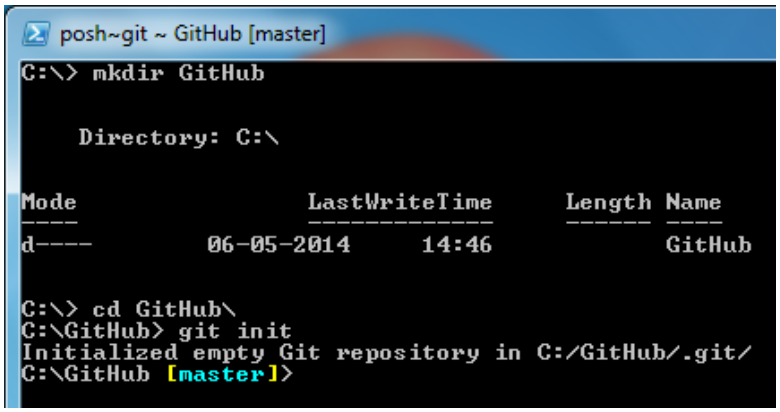
Comandos Git

Esta poderosa ferramenta integra diversos comandos que podem ser usados para configurar um utilizador, criar um repositório local, cometer modificações, alterações em grupo, remover e relocalizar ficheiros seguidos, excluir temporariamente ficheiros e diretorias, arquivar e restaurar alterações incompletas, listar e inspecionar a evolução dos ficheiros nos projetos, refazer alterações realizadas erradamente ou contendo erros e, por último, sincronizar as alterações efetuadas num repositório local com um servidor remoto.

Git practical playground

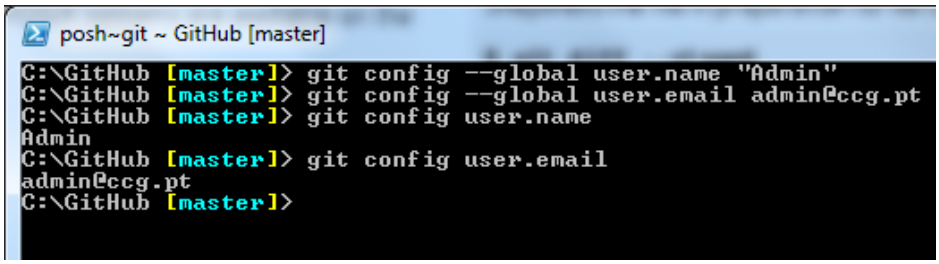
Neste capítulo foi executado um tutorial usando os comandos básicos deste DVCS. Assim conseguimos compreender melhor como funciona o Git, aplicando os conceitos teóricos adquiridos aliado à prática. Neste exemplo é usada a versão 1.9.2 do Git em Windows PowerShell mas poderá ser também usado o Git GUI ou Git bash.

1) Inicializar um repositório



Repositório Git inicializado e o *branch "master"* automaticamente criado e inicializado

git init

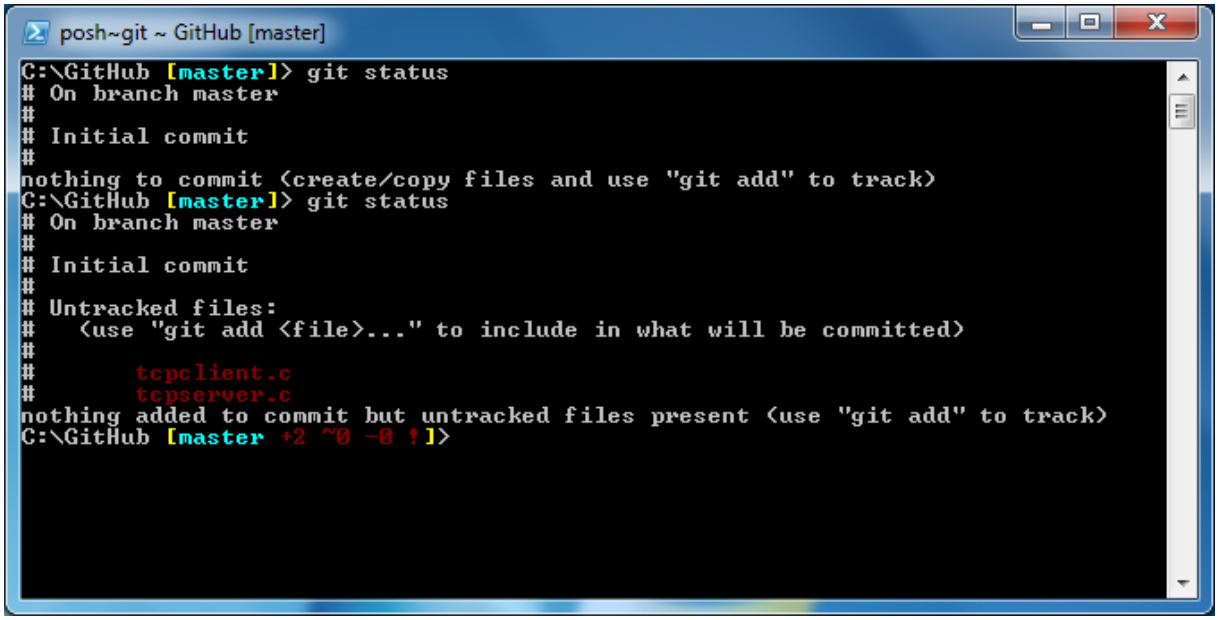


Configurações do utilizador Git. Bastante útil quando se está a trabalhar com projetos de equipa.

git config –global user.name "[name]"

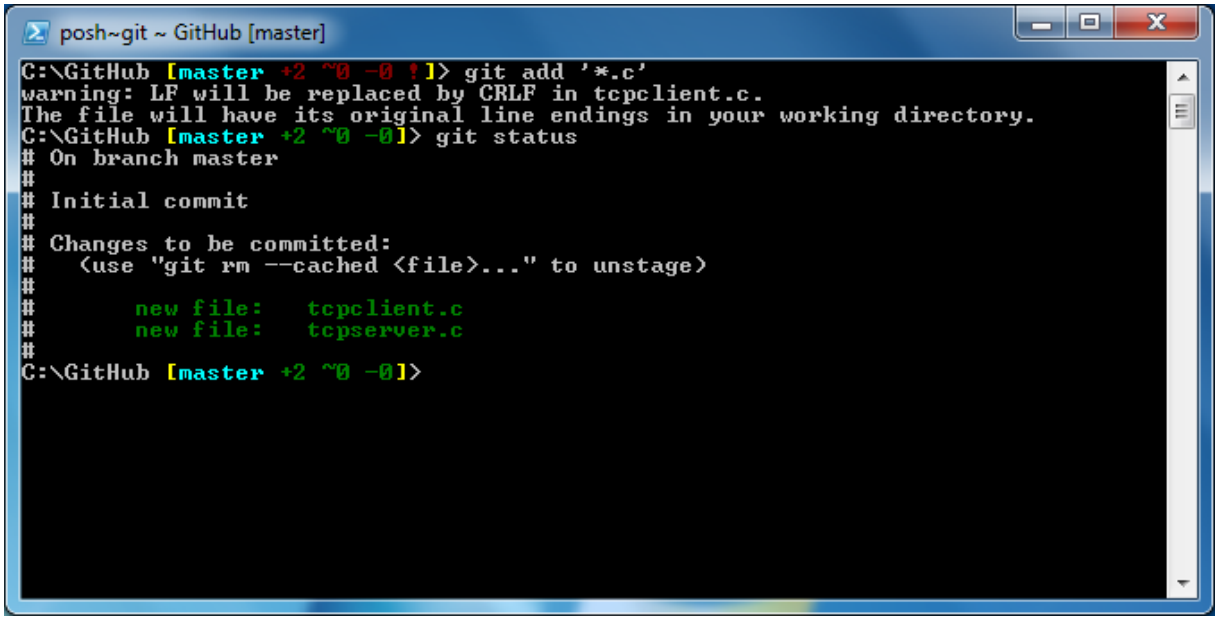
git config –global user.email [email]

2) Diretório de trabalho e *staging área*



O primeiro comando mostra que não existe nenhum ficheiro pronto para que se possa realizar um *commit*. Depois de serem copiados dois ficheiros para esse diretório, volta-se a executar o mesmo comando e assim o sistema de subversões deteta que existem dois ficheiros que não estão a ser seguidos pelo sistema (*untracked files)*.

git status

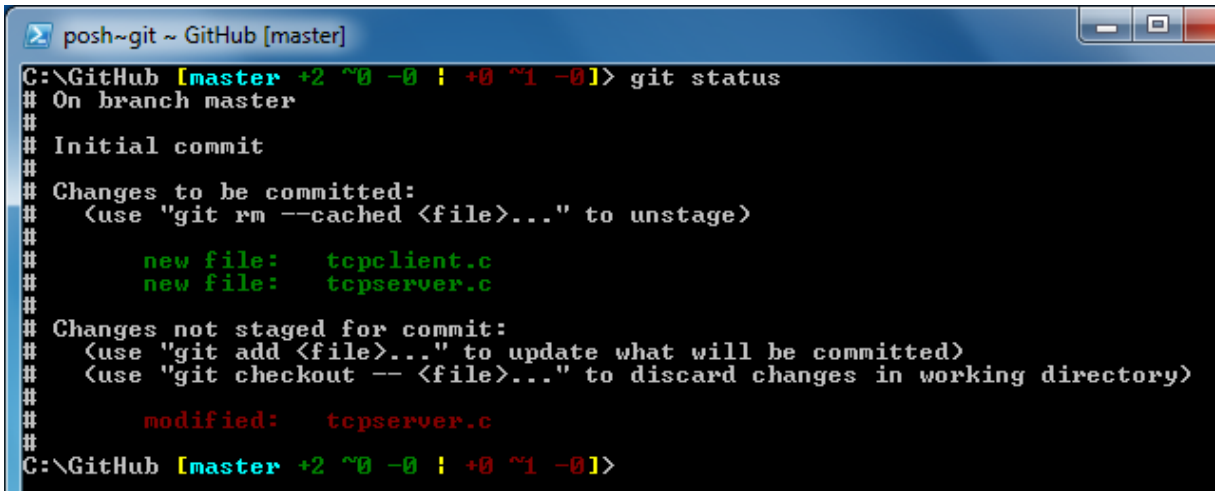


Para avisar o Git que este deve seguir as alterações realizadas a estes dois ficheiros copiados, temos então que executar o comando *add*. Estes ficheiros serão assim enviados para

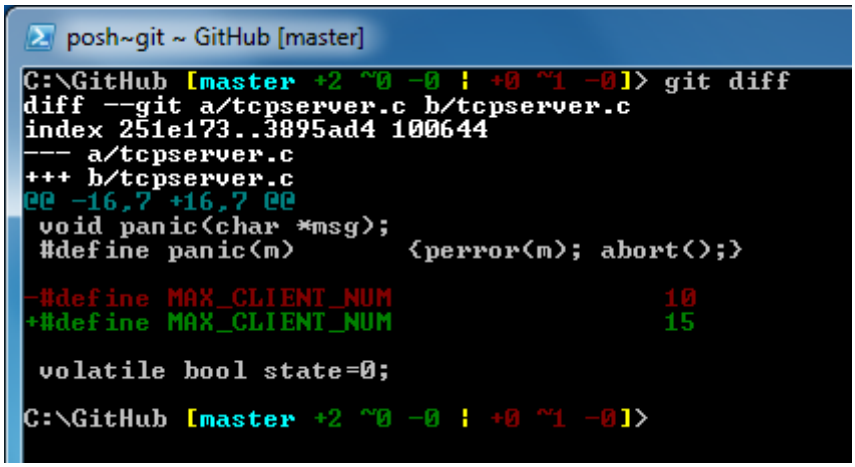
a chamada *staging area* onde a partir deste momento quaisquer modificações serão detetadas pelo nosso DVCS.

git add '*.c'

3) Staging area e *commit*

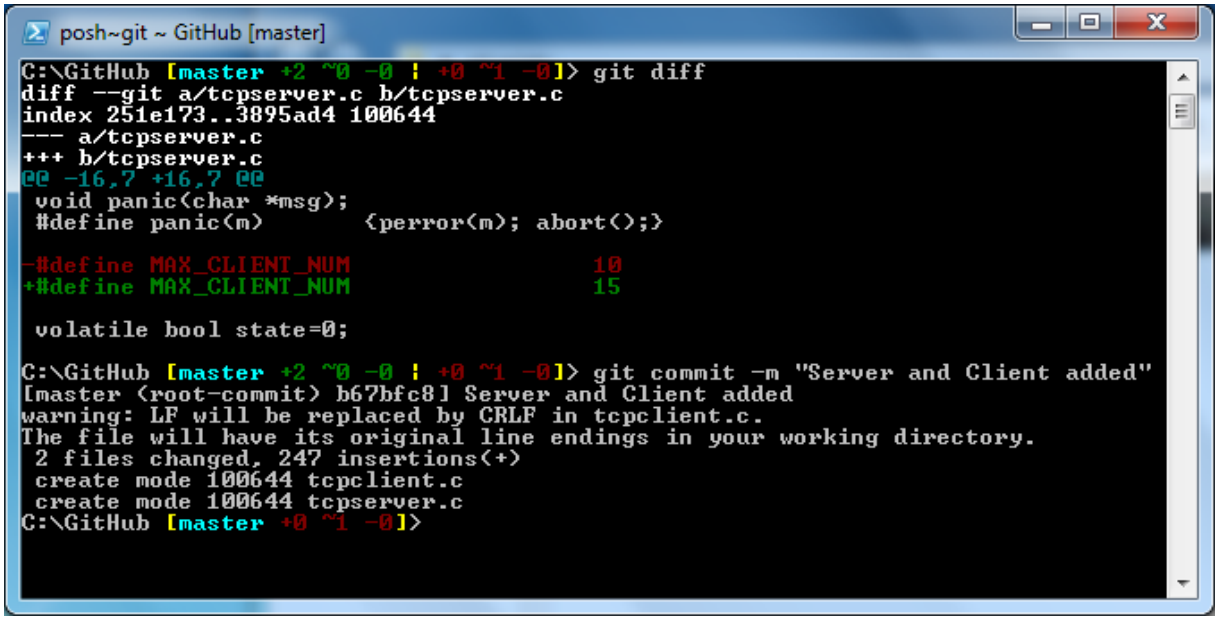


O Git, a partir de diferentes métodos explicados anteriormente, deteta diferenças entre modificações aos ficheiros que se encontram na *staging area*. Depois de alterar os ficheiros adicionados anteriormente ao sistema, estes foram então alterados e o comando *status* foi de seguida executado. O resultado deste comando inclui uma mensagem que avisa o utilizador que existem ficheiros que foram modificados e que este deverá proceder a um *commit* caso queira armazenar as modificações produzidas.



Para saber quais são as diferenças entre o ficheiro que foi inicialmente armazenado como *staged file* para a nova versão, executa-se o comando *diff*. O *output* deste comando mostra as diferenças em que a linha começando pelo caracter "+" (a verde) indica código que foi introduzido relativamente à versão anterior e a linha começando pelo "-" (a vermelho) revela o conteúdo retirado.

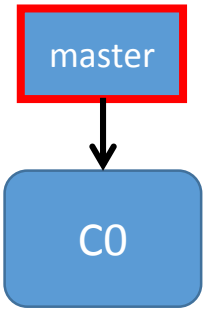
git diff



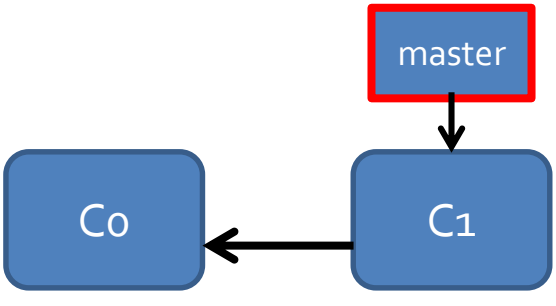
Cometer arquivos significa armazenar o seu conteúdo, naquele momento, num novo *commit* juntamente com uma mensagem de *log* opcional descrevendo as modificações efetuadas. A imagem em cima mostra como foi realizado este *commit* após se verificar a troca de um valor na macro MAX_CLIENT_NUM.

git commit –m [“message”]

Commit inicial – quando foram adicionados os ficheiros à *staging area*

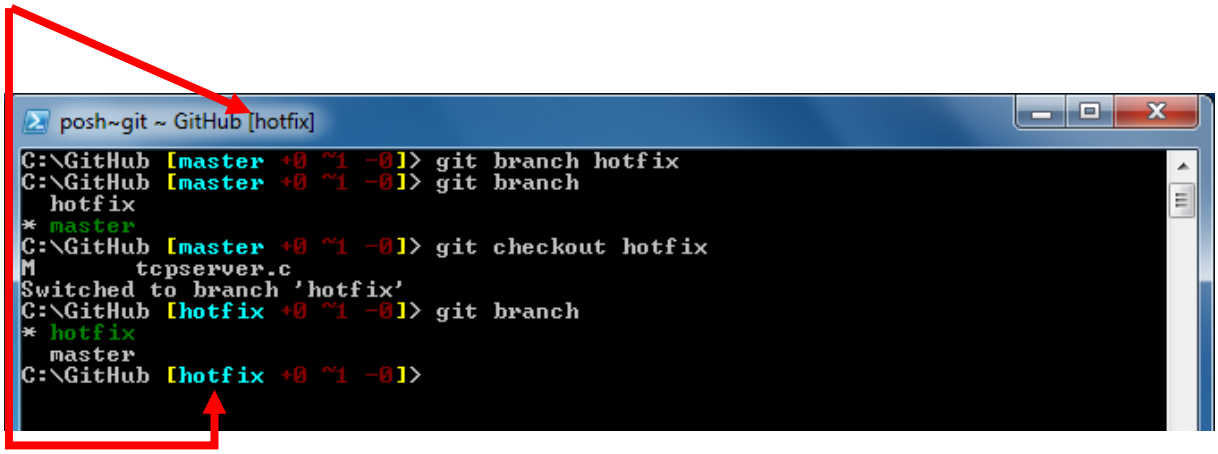


Novo *commit* – o *branch* “master” estará a apontar para o *commit* mais recente



4) Operações básicas com *branches* (*branching)*

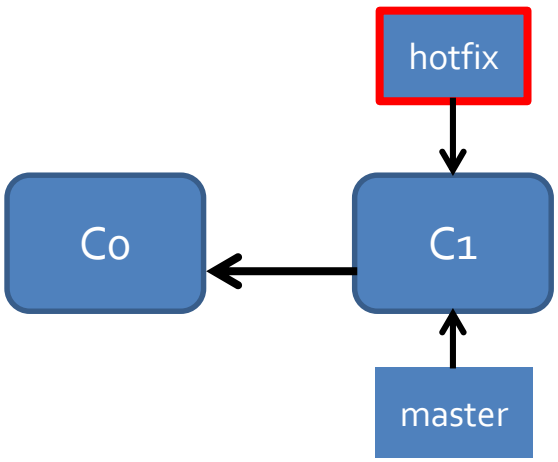
Um *branch* é um simples apontador móvel que se descola entre *commits*. O *branch* por defeito, criado automaticamente pelo Git, denomina-se por *master*. Ao passar do tempo, em função dos *commits*, este *branch* vai-se deslocando e apontando para o *commit* mais recente. O *branch* que o DVCS está a utilizar neste momento é representado por o nome entre parêntesis retos, à direita do caminho da diretoria.

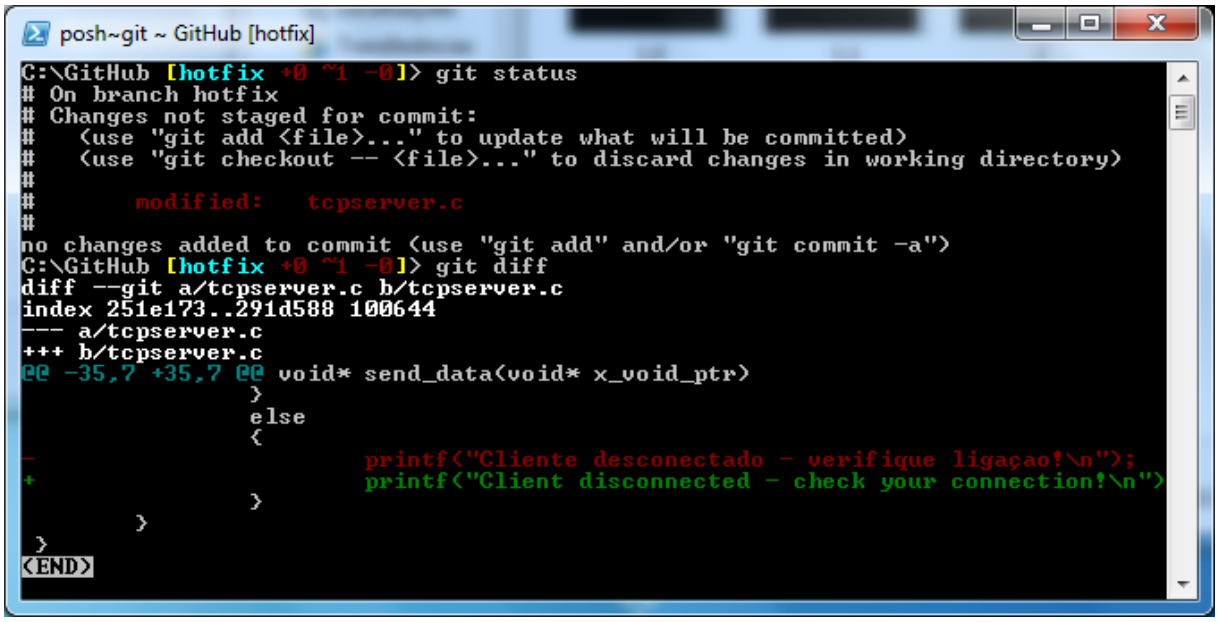


A primeira execução "git branch hotfix" faz com que seja criado um *branch* com o nome *hotfix*. Já a segunda linha "git branch" lista todos os "apontadores" disponíveis, sendo que o que está atualmente em uso é o *master branch* (asterisco, cor verde). Finalmente o comando "git checkout hotfix" faz com que o Git altere o *branch* em uso para o *hotfix*, fazendo com que todos os *commits* realizados a partir deste momento sejam ladeados pelo *hotfix branch*.

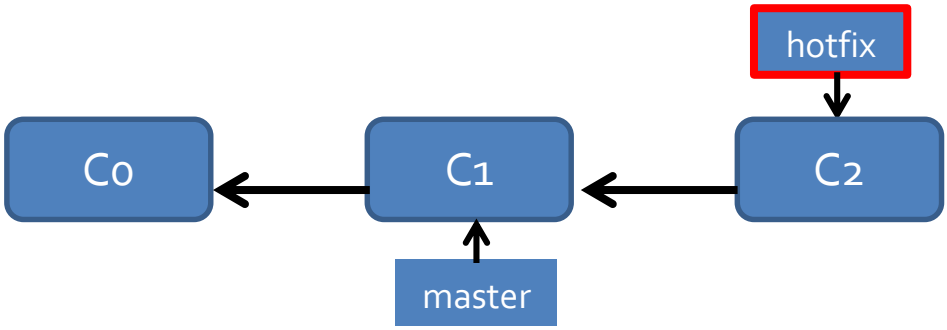
git branch

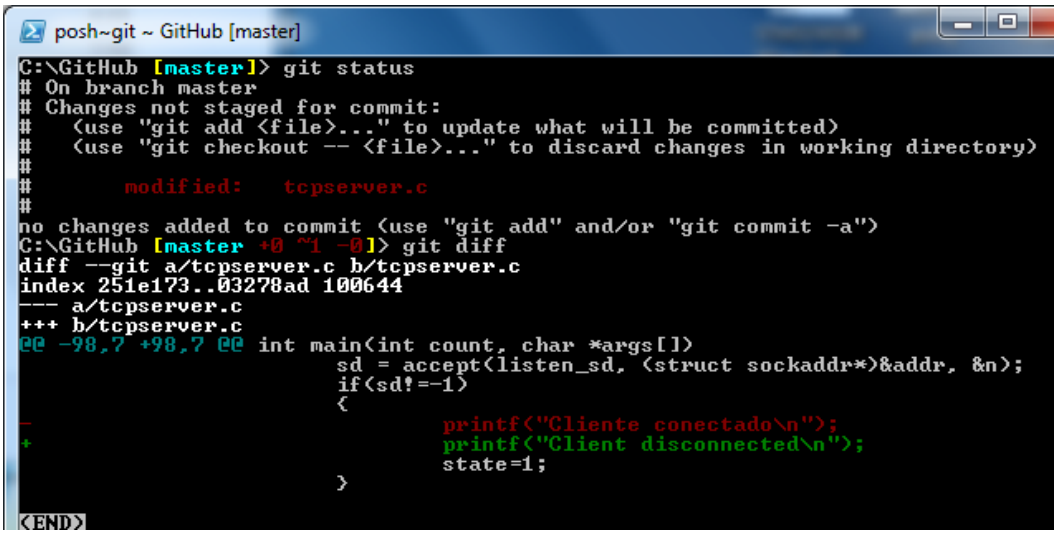
git checkout [branch name]



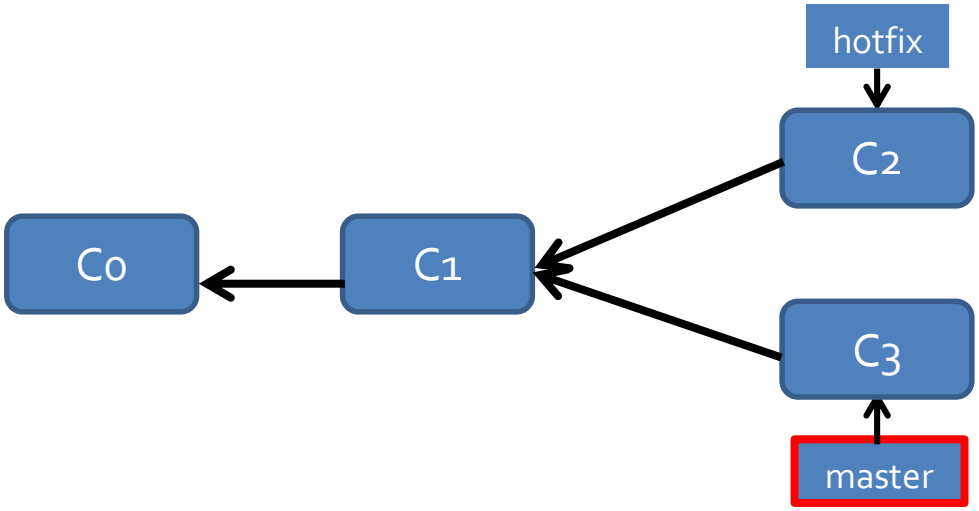


Imaginemos agora que algum tempo passou e que foram alterados os ficheiros do código fonte do projeto a desenvolver. O *commit* foi realizado com o *branch* atual, *hotfix*.



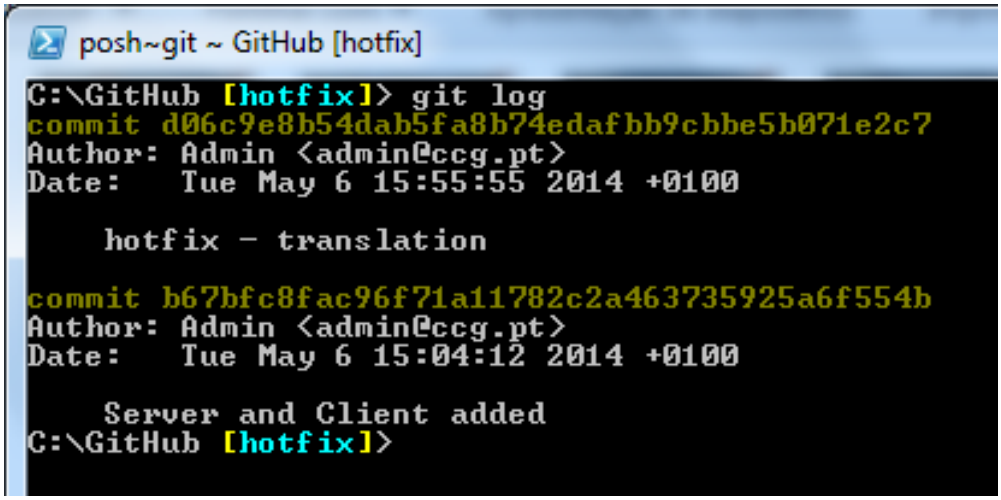


Foi realizada novamente uma alteração, mas desta vez no *branch* master.



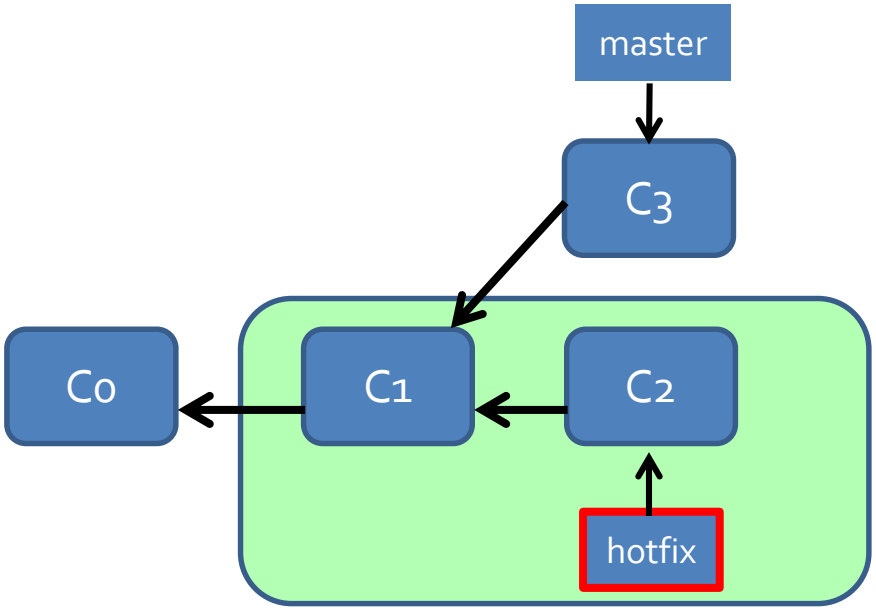
5) *Logs* e histórico de *branches* divergidos

Vão ser exibidos alguns métodos de como o sistema Git lista as alterações efetuadas e como é que essas alterações são diferenciadas.

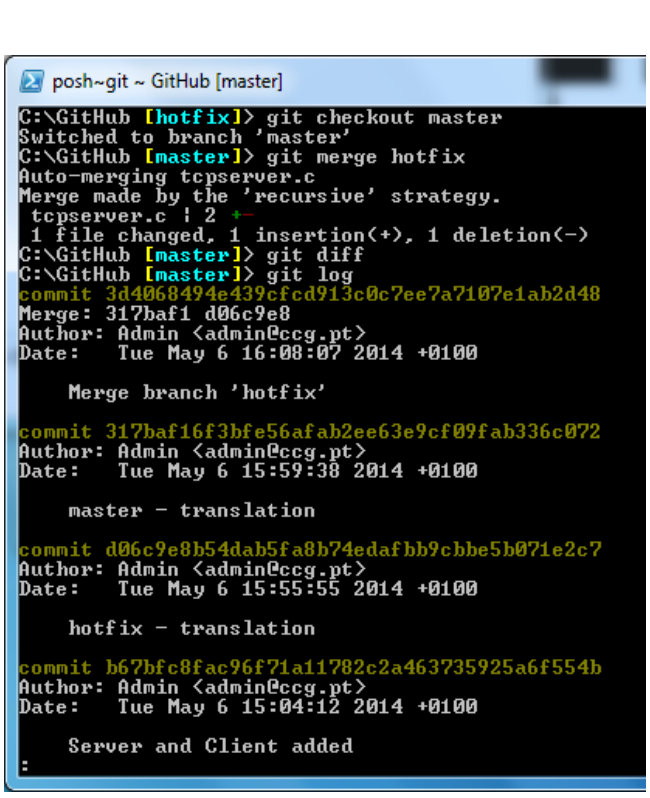


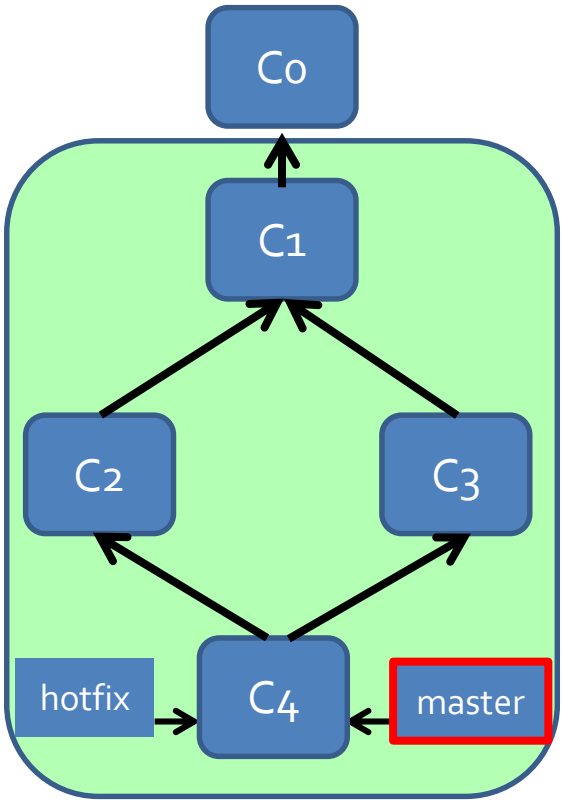
Com o *branch hotfix* selecionado é mostrado o historial de *commits* executados até ao momento em que o apontador *hotfix* está localizado. Juntamente com o autor, data e descrição do *commit* existe uma soma de verificação do conjunto de *snapshots*.

git log



6) Realizando o *merge* entre uma bifurcação e os seus *logs*





Neste exemplo é realizado um *merge* onde as modificações realizadas no *commit 3* e no *commit 2* são fundidas e cria-se então o *commit 4* onde ambos os *branches* estão a apontar para essa posição. Também é executado o comando "git log", mostrando o resultado da fusão.

```
git branch [branch name]
```

7) Apagar *branches*