Monitorização de parâmetros fisiológicos

André Oliveira
MIEEC, FEUP
up201405639@fe.up.pt

Baltasar Aroso
MIEEC, FEUP
up201404125@fe.up.pt

Renato Cruz
MIEEC, FEUP
up201405867@fe.up.pt

***Resumo*—Este projecto centrou-se no estudo de *middlewares*, mais especificamente OM2M, e a latência introduzida por este, procurando obter métodos eficazes capazes de medir a latência de comunicações em tempo real, de forma a compensar os atrasos introduzidos pela rede e pela camada protocolar deste *middleware*.**

A camada da aplicação pretendida, teve como objetivo a leitura e publicação de dados lidos pelo sensor MAX30100, para serem analisados numa base de dados, e utilizados posteriormente por serviços de saúde.

Toda a arquitetura necessária para a implementação deste projeto é apresentada neste documento, desde a comunicação entre o módulo ESP8266 e o sensor utilizados, até à comunicação ponto a ponto entre o ESP8266 e a aplicação de monitorização, através do *middleware* OM2M.

Na secção dos resultados encontra-se as medições obtidas, com referência aos valores máximos e mínimos das latências nas comunicações, bem como a visualização dos gráficos durante um espaço de observação.

I. Introdução

A constante investigação científica e a incessante competição por novas descobertas permitem cada vez mais responder às exigências de uma sociedade consumista e ansiosa por novas evoluções tecnológicas. O aumento do conhecimento em inúmeras áreas, como na saúde, perspetiva o aumento da qualidade de vida do ser humano e, consequentemente, o aumento da esperança média de vida do mesmo. A manutenção desta qualidade de vida exige uma maior comunicação entre pacientes e médicos, um conhecimento periódico e detalhado de fatores por vezes vitais na saúde dos utilizadores e um aumento da capacidade de resposta. Para tal, a monitorização de parâmetros fisiológicos remotamente é, hoje, vista como uma aplicação extremamente útil de tecnologias focadas na transmissão de informação, como o *Middleware*.

A conciliação de conceitos como *Machine to Machine* e *Internet of Things* remete para a monitorização quase instantânea de dados recolhidos por sensores usados pelos utilizadores em causa. A informação recolhida por estes sensores é passada para bases de dados para serem analisadas. Posteriormente, ações de prevenção e tratamento podem ser iniciadas mais rapidamente.

II. Definição do problema

Neste tipo de aplicações é essencial haver sincronização temporal entre os dados recolhidos pelos sensores e os dados inseridos nas bases de dados, já que o tempo associado às leituras é de extrema importância quando estão em jogo vidas humanas.

Para além disso, a fiabilidade no transporte da informação é outro aspeto a ter em consideração, uma vez que informação perdida pode levar à falta de prevenção ou tratamento, o que também coloca em risco a qualidade de vida dos utilizadores.

A utilização de uma interface gráfica para a análise da informação recolhida pelos sensores é outro ponto fulcral para que a monitorização dos parâmetros fisiológicos obtidos seja simples e eficiente.

Perante estes fatores é imprescindível que a performance do modo de transmissão de dados seja determinada para perceber a fiabilidade desta aplicação e o seu impacto num ambiente real. A escolha do *Middleware* implementado é uma das características que irá influenciar todos os aspetos referidos anteriormente.

III. Descrição da solução

A. Arquitetura

Para responder aos problemas de sincronização e armazenamento dos dados previamente enunciados, foi implementado um modelo de cooperação *Publisher-Subscriber*, assente no *middleware* OM2M (que implementa os standards *oneM2M* e *SmartM2M*) em que cada sensor (neste caso, um MAX30100) está ligado à rede através de um módulo ESP8266, que estará encarregue de comunicar com um *Broker*, assumindo assim o papel de *Publisher*. É necessário, ainda, um *Monitor* para receber os dados do sensor e tomar decisões, enquadrando-se nesta arquitetura como *Subscriber* dos dados gerados pelo(s) sensor(es).

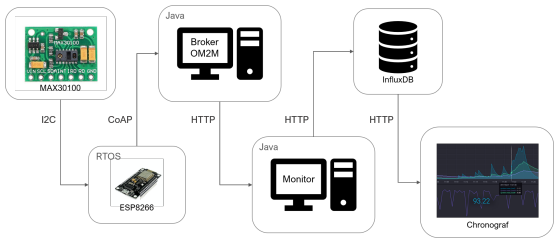

Figura 1. Arquitetura do sistema

Na Figura 1 encontra-se uma representação da arquitetura implementada, contendo os diferentes componentes utilizados e os respetivos ambientes usados para a sua implementação.

1) Comunicação com o Broker: Como referido anteriormente, foi utilizado um microcontrolador com capacidades de comunicação sem fios para a transmissão de dados recebidos pelo sensor MAX30100, o ESP8266. Este módulo sem fios

corre num sistema operativo RTOS (*Real-Time Operating System*), mais especificamente o sistema operativo *open-source FreeRTOS*. Já o sensor, responsável pela introdução de dados no sistema, suporta comunicações I2C (*Inter-Integrated Circuit*). Perante as enunciadas características dos dois componentes e de forma a garantir a correta comunicação entre eles, foi utilizada a biblioteca do *FreeRTOS* I2C e implementado o protocolo de comunicação especifico do MAX30100 [1]. Como não foi encontrada uma implementação deste protocolo usando a biblioteca I2C do *FreeRTOS*, foi necessário implementá-la de raiz, adaptando-a às necessidades deste projeto.

Após concluída a comunicação entre o sensor MAX30100 e o módulo ESP8266, foi necessário utilizar a biblioteca OM2M fornecida pelo laboratório DaRTES (*Distributed and Real-Time Embedded Systems*) da Faculdade de Engenharia da Universidade do Porto de modo a estabelecer a comunicação com o *Broker* e publicar os dados do sensor.

2) Broker: A comunicação entre o módulo ESP8266 e o *Broker* é efetuada recorrendo ao protocolo de transporte CoAP (*Constrained Application Protocol*). Este protocolo é designado para aplicações *Machine to Machine* e fornece um modelo de interação *request/response* entre aplicações. Neste protocolo, um pedido é efetuado de modo confirmável (*CON*) ou não confirmável (*NON*). Caso o destinatário (*Broker*) esteja disponível e o pedido seja do tipo *CON*, o mesmo é replicado e retornado no *Acknowledgement (ACK)* (*piggybacked*). Deste modo, é possível garantir a fiabilidade da transmissão dos dados, um problema anteriormente mencionado. Para além disso, os dados trocados através deste protocolo podem estar em vários formatos, sendo que há uma restrição entre *XML* ou *JSON* imposta pelo uso do OM2M[1]. Nesta implementação, optou-se pelo uso de *JSON*, já que este produz um *overhead* menos significativo na representação e leitura dos dados.

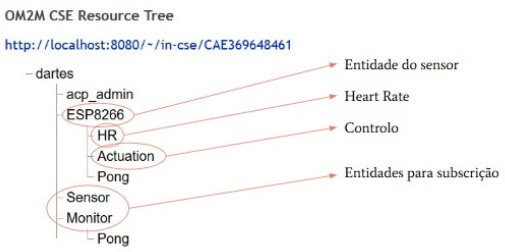

Figura 2. Arquitetura do *Broker* para publicações

Na Figura 2 pode-se observar a estrutura utilizada para as publicações e subsequentes subscrições. Para a troca de dados e controlo do sensor, foi criada uma *Aplication Enitity*, denominada *ESP8266*, que contém dois *Containers*: *HR (Heart Rate)* e *Actuation*. Estes *containers* servem para o sensor publicar leituras e o *Monitor* controlar o sensor, respetivamente. As entidades *Sensor* e *Monitor* servem para cada um fazer subscrições pertinentes aos tópicos, *Actuation* e *HR* respetivamente.

O módulo ESP8266 está encarregue da criação de todas as entidades e *containers* pertinentes, pelo que ao *Monitor* resta subscrever aos dados publicados em *ESP8266/HR*. Para tal, começa por criar a entidade *Monitor* e, de seguida, efetua a subscrição. Se aplicável, pode ainda publicar conteúdos de controlo para o container *ESP8266/Actuation*, como referido anteriomente.

Deve ainda ser referido que tanto o *Broker* como o *Monitor* correm sobre uma *Java Virtual Machine*, visto serem aplicações desenvolvidas em Java.

3) Monitorização dos dados: Tanto a comunicação entre o *Broker* e o *Monitor* como a comunicação entre o *Monitor* e o *InfluxDB* é feita através de pedidos *HTTP*.

Como referido anteriormente, o *Monitor* é definido como o único *subscriber* que receberá informação de vários *publishers*. Para armazenar e facilitar a visualização dos dados recebidos, foi utilizada uma base de dados não relacional, devido à natureza e capacidade deste tipo de bases de dados ao lidar com grandes quantidades de informação. Mais concretamente, optou-se pelo *InfluxDB*, uma base de dados *open-source* orientada a eventos temporais (*Time Series Database*). Como interface gráfica usou-se o *Chronograf*, dada a sua facilidade de integração com o InfluxDB. A comunicação entre esta base de dados e a interface gráfica é feita com recurso ao protocolo HTTP.

B. Medições

Neste subsecção descreve-se as duas medições efetuadas: latências parciais - entre o ESP8266 e o *Broker*, e entre o *Broker* e o *Monitor* - e latência total ponto a ponto - entre o ESP8266 e o *Monitor*. A latência ponto a ponto, para além das latências parciais, inclui o eventual tempo de processamento, nomeadamente no *Broker*: desde que o conteúdo é publicado, até que é enviado para o *Monitor*.

1) Latências parciais: Para se efetuar medições das latências parciais, criou-se dois *containers* adicionais: os *containers Pong*. Estes *containers* têm como objetivo receber periodicamente publicações com *timestamps* medidos no ponto onde se quer saber a latência, seja o ESP8266 ou o *Monitor*, tal como demonstrado na Figura 3. A partir da Figura 4 pode-se observar o método utilizado: a cada publicação do

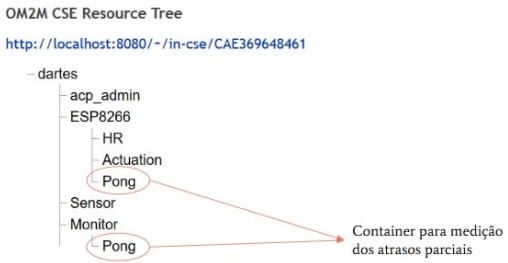

Figura 3. Arquitetura do *Broker* para medições

[1]https://www.eclipse.org/om2m/

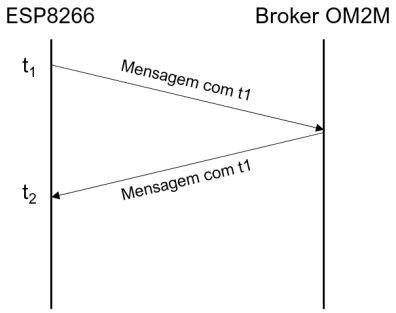

Figura 4. Medição de atraso das comunicações

timestamp, o *Broker* devolve uma confirmação do conteúdo. Esta confirmação contém o *timestamp* enviado, pelas razões mencionadas anteriormente em III-A2. A partir desta mensagem de confirmação calcula-se a diferença de tempos entre o *timestamp* enviado e o tempo de receção desta mensagem, obtendo-se assim uma estimativa do *Round Trip Time* (RTT) e, consequentemente, do atraso introduzido nas comunicações:

$$atraso = \frac{t_2 - t_1}{2} \tag{1}$$

2) Latência ponto a ponto: Com os tempos parciais medidos, resta medir a latência entre os pontos terminais: ESP8266 e *Monitor*.
Esta medição tem dois objetivos principais:

- Estimação do tempo de processamento do *Broker*
- Estimação da *performance* do *middleware* OM2M

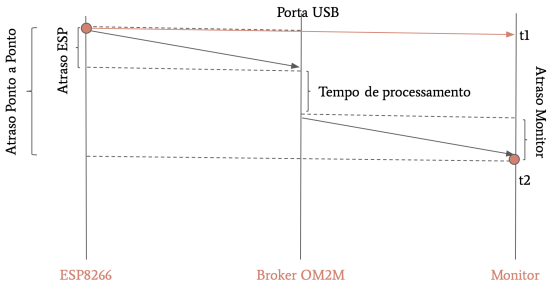

Figura 5. Medição de atraso ponto a ponto (e2e)

Na Figura 5 observa-se o método utilizado para medir a latência ponto a ponto: entre o ESP8266 e o *Monitor*. Desprezando o tempo de comunicação USB-Serial, ligou-se o ESP8266 à máquina responsável por desempenhar as funções de *Monitor*. Quando o ESP8266 efetua uma publicação de valores medidos pelo sensor, envia uma mensagem com o *resource name* da publicação para a porta USB, podendo ser captada pelo *Monitor*. Este, por sua vez, grava o *timestamp* (t_1) de receção desta mensagem associado ao *resource name*. Quando este recebe a notificação da publicação (através da subscrição) do *resource name* correspondente, guarda um novo *timestamp* (t_2). Por último, faz a diferença entre os dois *timestamps*, obtendo assim o tempo total que uma publicação demora até chegar ao subscritor, desde a sua criação.

$$atraso_{e2e} = t2 - t1 \tag{2}$$

Com este tempo calculado, e usando os tempos parciais calculados anteriormente, é possível estimar o tempo de processamento do *Broker*. Além disto, sabe-se também que para efetuar todo o procedimento de publicações/notificações, diversas mensagens protocolares, como *acknowledges*, são trocadas entre os participantes e o *Broker*. Este tempo calculado engloba, portanto, o atraso introduzido por estes procedimentos, e torna possível obter a *performance* do *middleware* OM2M.

IV. Resultados

A. Medições

Tabela I
Máximos e mínimos das latências medidas

	Ponto a Ponto	ESP8266 ↔ Broker	Broker ↔ Monitor
Máximo	393 061 354 ns	129 630 351 ns	167 153 858 ns
Mínimo	2 032 125 ns	15 985 615 ns	12 024 579 ns

É importante referir que, embora os dados na Tabela I estejam representados em nanossegundos, não há garantia de que esteja seja a resolução efetiva das funções utilizadas para o efeito.

No módulo ESP8266, as medições estão sujeitas a uma frequência de processador relativamente baixa, proporcionando assim uma precisão na ordem dos microssegundos. Para efeitos de medição baseada na diferença de valores, a precisão desempenha um papel fundamental relativamente à exatidão, visto que os valores podem ter um referencial arbitrário, sendo apenas desejável um relógio monotónico.

No caso da linguagem Java, as medições foram efetuadas recorrendo à função *System.nanoTime()*, a qual apresenta precisão de nanossegundo, mas não necessariamente resolução de nanossegundo [7]. A resolução pode variar bastante consoante o sistema e o hardware em que se encontra o programa, assim como o número de *threads* que o programa está a correr de momento. Assim, no melhor caso, a função pode oferecer uma resolução na ordem dos 30 ns. Nos piores casos, pode chegar além dos 15 µs.[2]

É ainda importante mencionar o facto de que a latência ponto a ponto pode apresentar valores inferiores à soma das latências das duas ligações intermédias: isto acontece devido ao valor das latências intermédias ser atualizado com base numa média ajustável, segundo a equação:

$$media_{nova} = (1-\alpha) \times media_{antiga} + \alpha \times atraso_{medido} \tag{3}$$

Em contraste, o atraso ponto a ponto é medido e registado **por publicação**, sendo que este é bastante mais fiável como medida de performance.

[2]https://shipilev.net/blog/2014/nanotrusting-nanotime/ [Acedido em 17 Jan. 2019]

B. Visualização

Na Figura 6, é possível observar uma das principais ferramentas do *Chronograf*, a disposição de tabelas retornadas através de *queries* construídas pela interface gráfica, proporcionando uma fácil e versátil visualização dos dados, assim como a aplicabilidade de funções de conveniência (médias, somas, contagens, ...).

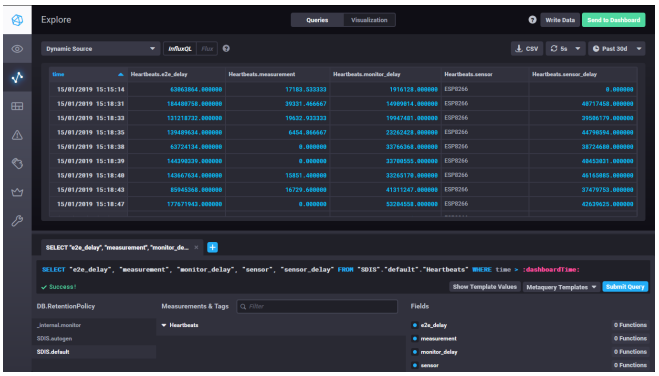

Figura 6. Resultado de uma *query* efetuada automaticamente à base de dados

Para além disso, é possível visualizar os dados diretamente na interface gráfica (Figura 8) ou exportar os dados em formato Comma Separated Values (CSV) (Figura 7) de modo a permitir o tratamento dos dados num outro software de eleição.

Nas Figuras 7 e 8, podemos observar 3 períodos: um inicial de pouco congestionamento na rede; um período intermédio em que os valores são superiores e mais espaçados, indicando que a rede entrou em congestionamento, e portanto havendo uma maior duração na transição dos dados entre o sensor e o *Monitor*; um último período em que a rede volta a entrar em pouco congestionamento, voltando os atrasos ponto a ponto a diminuir e tendo os eventos ocorrido em intervalos mais curtos.

V. Avaliação da complexidade

A. Resumo

Durante a elaboração deste projeto encontrou-se diversos problemas, expostos nas alíneas seguintes, os quais exigiram um trabalho acrescido na sua deteção, pesquisa de soluções adequadas e respetivas resoluções. Este trabalho adicional e inesperado encurtou o tempo disponível para ser dedicado aos objetivos principais do projeto, inicialmente definidos.

B. Complexidade da arquitetura

Começando pela comunicação com o sensor MAX30100, como não havia qualquer suporte para o protocolo específico, o desenvolvimento de uma biblioteca capaz de ler/escrever e comunicar com o sensor tornou-se um ponto principal na fase inicial do projeto. A biblioteca desenvolvida encontra-se capaz de ler e escrever qualquer registo do sensor, no entanto e devido a algumas questões quanto aos protocolos enunciados na *datasheet* deste sensor, nomeadamente relativas à leitura do FIFO de dados, não se concluiu na totalidade este ponto, e por isso os dados apresentados sobre os batimentos cardíacos não estão completamente corretos. Este ponto, ainda que interessante para questões de visualização, foi colocado em segundo plano e a prova de conceito da visualização conseguiu-se dispondo das medidas de latência.

Continuando com a implementação da proposta anteriormente explicada, após criadas as subscrições e analisando capturas do *software Wireshark*, observou-se que notificações de novas publicações eram comunicadas ao ESP8266 através de pedidos POST. Analisando a fundo a biblioteca do *FreeRTOS*, notou-se uma falta de métodos para tratar estes pedidos, impossibilitando o processamento de publicações no *container Actuation*, impedindo assim a tomada de ações sobre o módulo. Para resolver esta situação de grande relevo, optou-se por modificar a biblioteca *net.h* do *FreeRTOS* de forma a que fosse possível escolher de que forma pedidos HTTP eram processados:

- Novo cabeçalho para um novo tipo de função (*coap_request_handler_t*).
- Acrescentou-se à estrutura *coap_context_t* este novo tipo de função.
- Nova função *coap_register_request_handler* para registar o *handler* pretendido ao *context* em uso.

Por útlimo, para acrescentar esta nova funcionalidade, foi necessário modificar o ficheiro *net.c*: na função *handle_request*, onde qualquer pedido HTTP válido era respondido com um simples HTTP 205 (Reset Content), realizou-se uma chamada à função previamente registada (*context→request_handler(context,request)*). Depois desta funcionalidade implementada, é apenas necessário ao utilizador criar uma função do tipo *coap_request_handler_t*, onde se processam os pedidos HTTP, e registá-la com uma chamada a *coap_register_request_handler*.

C. Complexidade das medições

Tal como exposto anteriormente, para se efetuar as medições de latências parciais, foi necessário fazer as modificações indicadas na Figura 3. Estas alterações tiveram impacto tanto no código do ESP8266 como no código do *Monitor*, sendo necessário elaborar um algoritmo de *probing* capaz de periodicamente estimar a latência das comunicações dos dois lados do *Broker*. A criação de novos *containers*, uso de *threads* para o referido *probing*, leitura das respostas do *Broker* e por último o calculo da latência foram todos esforços não desprezavéis.

Para o cálculo de latências totais (ponto a ponto), foi mais uma vez necessário efetuar alterações no código do *Monitor*, introduzindo uma nova biblioteca para comunicações USB-Serial [3].

D. Complexidade da visualização

Por último, foi necessário visualizar os dados, tanto das medições do sensor, como das latências calculadas, Tal como referido anteriormente, foi utilizada uma base de dados não-relacional, *InfluxDB*, e a interface gráfica associada, *Chronograf*. O uso desta base de dados implicou mais alterações ao código do *Monitor*, para acrescentar suporte a esta base de

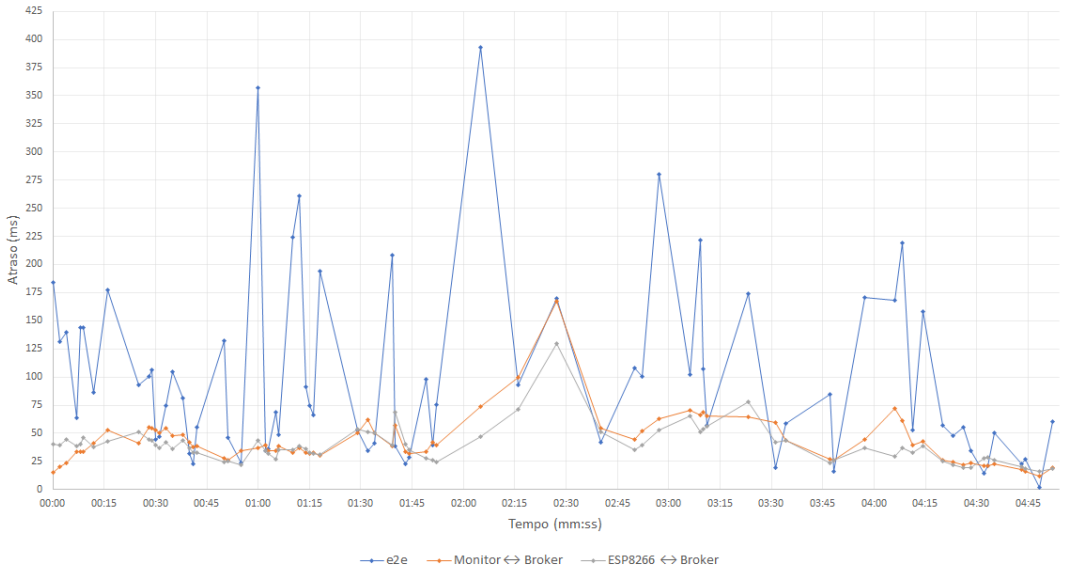

Figura 7. Resultados obtidos na medição de latências (Excel)

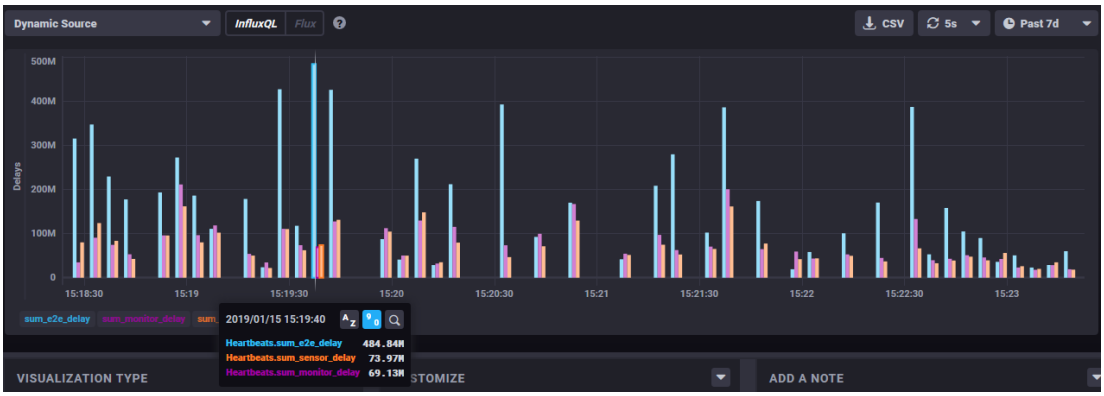

Figura 8. Resultados obtidos na medição de latências (Chronograf)

dados, facilitando as *queries* para introdução de novos dados [4]. Foi também necessário instalar e configurar a base de dados e a interface gráfica de modo a interagir com o código Java após hospedados numa máquina ([5], [6]).

VI. Auto-Avaliação

Durante todo o projeto o grupo discutiu ativamente propostas de implementação e de resolução dos problemas que foram aparecendo. De referir também que todo o trabalho foi realizado em conjunto e com participação pro-ativa dos três membros.

André Oliveira – 40% – Modificações do *Monitor*: integração do *InfluxDB*, *Chronograf*, medição de latência parcial do *Monitor*, ajuda na comunicação USB-Serial para latências totais (lado do *Monitor*). Desenvolvimento dos algoritmos de latências.

Baltasar Aroso – 20% – Participação no desenvolvimento dos algoritmos para medições de latência, ajuda na resolução de alguns dos problemas. Elaboração da apresentação.

Renato Cruz – 40% – Comunicação com o sensor MAX30100. Implementação do código do ESP8266: Comunicação com o *Broker* (publicações/subscrições), medidas de latências parciais do sensor, e total (lado do sensor). Modificações na biblioteca *FreeRTOS*. Desenvolvimento dos algoritmos de latências.

Acknowledgment

Este projeto está enquadrado na unidade curricular SDIS, sendo o professor responsável pela orientação do mesmo o Professor Luís Almeida. Ao longo de toda a implementação do projeto houve um apoio incondicional e essencial do laboratório DaRTES, em particular da investigadora Diana Guimarães, essencialmente para a resolução dos problemas iniciais referidos como imprevistos. Para além disso, é de destacar o agradecimento ao professor por toda a disponibilidade que dispôs para que o projeto fosse concluído com sucesso.

REFERÊNCIAS

[1] MAX30100, Pulse Oximeter and Heart-Rate Sensor IC for Wearable Health, https://datasheets.maximintegrated.com/en/ds/MAX30100.pdf

[2] Tools.ietf.org. (2019). RFC 7252 - The Constrained Application Protocol (CoAP). [online] Available at: https://tools.ietf.org/html/rfc7252#page-12 [Acedido em 17 Jan. 2019]

[3] Java RXTX, http://rxtx.qbang.org/wiki/index.php/Main_Page

[4] Java InfluxDB, https://search.maven.org/classic

[5] InfluxDB, Key Features, https://docs.influxdata.com/influxdb/v1.7

[6] Chronograf, https://docs.influxdata.com/chronograf/v1.7/

[7] System (Java Platform SE 8), https://docs.oracle.com/javase/8/docs/api/java/lang/System.html#nanoTime-- [Acedido em 17 Jan. 2019]