PROPOSTA DE FRAMEWORK PARA DISPOSITIVOS DE TELEMETRIA ROBUSTOS E DE BAIXO CONSUMO BASEADOS EM IOT

Sayonara A. C. Tavares*, Danielly C. M. Costa*, Diego R. C. Silva*, Marcelo B. Nogueira*, Marconi C. Rodrigues*

*Escola de Ciências e Tecnologia - UFRN - Lagoa Nova
Natal, RN, Brasil

Emails: sayonaracirilo@bct.ect.ufrn.br, daniellycosta97@ufrn.edu.br, diego@ect.ufrn.br, marcelonogueira@ect.ufrn.br, marconicamara@ect.ufrn.br

Abstract— The present work proposes a set of solutions to the challenges of communication, connection management, memory, exceptions, synchronization and data persistence for the development of hardware devices that make up distributed systems over the Internet (IoT) and need to communicate with other components of the system. The hardware used in the proposal is described, as well as the algorithms that make the device robust. In the end, the results obtained in real applications as the hardware EEPROM lifespan extension due to circular record and data persistence because of a data structure used in a real application are shown and in the final considerations, the energy consumption is discussed.

Keywords— Internet of Things, telemetry, energy efficient, ESP8266.

Resumo— O presente trabalho propõe um conjunto de soluções à desafios de comunicação, gerenciamento de conexão, memória, exceções, sincronia e persistência de dados para o desenvolvimento de dispositivos de hardware que compõe sistemas distribuídos pela Internet (IoT) e precisam se comunicar com outros componentes do mesmo. Descreve-se o hardware utilizado na proposta e também os algoritmos que tornam o dispositivo robusto. Ao final mostra-se os resultados obtidos como: o prolongamento da vida útil da EEMPROM do hardware com a implementação de uma gravação circular e a persistência dos dados no hardware com a utilização de uma estrutura de dados em aplicações reais. Além disso, nas considerações finais comenta-se sobre o consumo de energia.

Palavras-chave— Internet das coisas, telemetria, baixo consumo, ESP8266

1 Introdução

Os mais recentes avanços tecnológicos tem permitido um número de aplicações sem precedentes. A IoT (internet of Things) ou Internet das Coisas (Zhang et al., 2012), e toda a expectativa de revolução inerente a ela, atua com um fim que tem direcionado o desenvolvimento de novos dispositivos com características extremamente atrativas para aplicações para o consumo. A eletrônica digital, as tecnologias de comunicação e informação (ICT - *Information and Communication Technologies*) (Azarov and Maiboroda, 2017) e os serviços de computação em nuvem têm mostrado uma posição essencial no surgimento de novos produtos, novos modelos de negócios, e desde já apontam para uma nova maneira de como o consumidor irá interagir com a tecnologia.

Dentre as várias possibilidades de novos produtos para o consumidor, como consequência dessa corrida tecnológica, estão a automação, há muito tempo empregada na indústria, de tarefas corriqueiras, em prédios comerciais e domiciliares; a monitoração do funcionamento e localização de veículos em tempo real; a automação de culturas de alimentos em uma maior escala e penetração; o aumento da segurança, etc. Isso sem falar em soluções de maior porte como setores públicos (*Smart Cities* (Albino et al., 2015; Zhuhadar et al., 2017)) que trazem ganhos para todos os cidadãos.

Para o funcionamento desses serviços modernos, dentre outras coisas, é necessário o desenvolvimento de dispositivos que funcionam na ponta (*Edge Computing* (Satyanarayanan, 2017; Ahmed and Rehmani, 2017)) executando tarefas, acionando atuadores e fazendo medições, através da conexão com os outros componentes do sistema. O desenvolvimento desses dispositivos apresentam alguns desafios de robustez e consumo, dentre eles, pode-se destacar: o gerenciamento da conexão de rede e suas consequências, como o envio de medições de sensores e o comportamento do dispositivo quando perde a conexão; o controle e monitoramento do consumo da memória; o tipo de comunicação e seus parâmetros; o acompanhamento das exceções que ocorrem nos dispositivos já em modo de produção; a perda de informações ao se reiniciar o dispositivo; a sincronização da data e da hora, para registro das medições, o gerenciamento de bateria, quando for o caso, etc.

Considerando tais desafios, faz-se necessário o desenvolvimento de dispositivos robustos, com capacidade de lidar com possíveis adversidades, como perda de conexão, sem que isso signifique necessariamente perda de dados, falta de fornecimento de energia elétrica, sem acarretar perda de configuração ou de dados, possibilidade de acompanhamento do estado interno do dispositivo como carga da bateria, nível do sinal, consumo de memória, etc, e de baixo consumo, para que não onerem a energia consumida ou que maximize o seu tempo de funcionamento quando alimentado

por bateria.

Nesse contexto, esse artigo propõe um conjunto de soluções para problemas comuns no desenvolvimento de dispositivos de hardware que precisam interagir, enviando dados e recebendo comandos, com sistemas distribuídos através da Internet, que constituem a arquitetura dos serviços de IoT.

O restante do artigo está organizado da seguinte maneira; A seção 2 descreve o hardware usado no desenvolvimento da proposta assim como as soluções de software capazes de tornar o dispositivo robusto e superar as adversidades já citadas. A seção 3, mostra alguns resultados já alcançados pela proposta e a seção 4, finalmente, traz algumas conclusões verificadas ao longo do desenvolvimento e ao se analisar os resultados práticos.

2 Metodologia

Muito se fala sobre o desenvolvimento de dispositivos para IoT. Porém, deve-se salientar que estes dispositivos são apenas parte de um sistema maior. Por definição, tais dispositivos são conectados à Internet consistindo apenas de um nó final que interage com o usuário ou com o ambiente. Para disponibilização dos dados coletados por tais dispositivos necessita-se de um servidor, geralmente fazendo uso de tecnologias de *cloud computing* (computação em nuvem) que armazena, processa e disponibiliza-os para os usuários. Estes, por sua vez, interagem com o sistema através de aplicativos computacionais que mostram os dados e servem de intermediários para a interação entre usuários e dispositivos (Soliman et al., 2013). A Figura 1 mostra o ambiente em que os dispositivos à que se refere este trabalho estão inseridos.

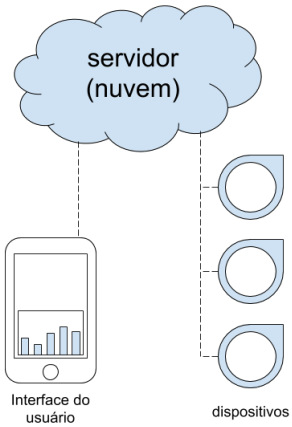

Figura 1: Arquitetura de sistemas de IoT.

Este trabalho trata especificamente de um framework de software para o desenvolvimento de dispositivos para serem inseridos neste tipo de ambiente. Ou seja, eles devem ser capazes de se comunicar com um servidor na internet e receber comandos de usuários ou do próprio sistema. Além disso, existem algumas característica desejáveis para que os usuários tenham uma boa experiência de uso. São elas:

- Alta disponibilidade;
- Robustez;
- Comunicação bidirecional;
- baixa latência de comunicação;
- Sincronização de relógios;
- Baixo consumo de energia;
- Persistência local de dados;

Ao utilizar a arquitetura proposta, o desenvolvedor pode definir várias características do dispositivo através da escolha de parâmetros de configuração. Dentre elas, o número e o tipo de sensores conectados ao dispositivo e o tipo de comunicação com o servidor, podendo ser síncrona ou assíncrona, e seus parâmetros específicos.

Neste trabalho, o *hardware* utilizado como base para a construção de dispositivos consiste de uma placa de prototipagem baseada no microcontrolador ESP8266 (ver Figura 2), possuindo entre suas vantagens compatibilidade com o padrão de comunicação 802.11 b/g/n(WiFi) de forma nativa, memória RAM de $80Kbytes$, Flash de $4Mbytes$ e um baixo custo comparado com outros concorrentes como o Arduíno (McRoberts, 2011), que apresenta desempenho e memória menor, além de não possuir WiFi integrado. Ademais, o ESP8266 possui portas que trabalham com a tecnologia de interrupções, tecnologia esta que pode ser usada para diferentes propósitos.(WEMOS Electronics, 2018; Espressif, 2018).

Figura 2: Wemos D1 mini, fonte: wiki.wemos.cc

As próximas subseções descrevem partes da implementação responsáveis pelo gerenciamento dos dados no que diz respeito a disponibilidade de comunicação, gasto de energia e falta de energia.

2.1 estrutura de dados - fila

Em aplicações que não requerem uma taxa de atualização de dados muito alta, manter uma conexão com o servidor aberta para envio de dados

acaba sendo muito custoso no que diz respeito ao consumo de energia e banda de comunicação. De modo a solucionar esse problema, foi implementada uma fila na memória RAM do dispositivo onde os dados provenientes dos sensores são armazenados. Após um intervalo de tempo pré definido, todo o conteúdo da fila é enviado ao servidor pelo dispositivo, que ao receber uma mensagem de confirmação de recebimento por parte do servidor, remove os itens da fila e disponibiliza àquela área na memória, para que todo o processo descrito torne a ser realizado.

Outra vantagem no desenvolvimento de tal estrutura para o armazenamento de dados, é que, em caso de falta de rede, identificado com o não recebimento da confirmação do servidor, o dispositivo continua depositando os valores medidos na fila, que terá seu conteúdo enviado após o retorno da rede. Além disso, com a necessidade de depuração da aplicação, uma estrutura do mesmo tipo pode ser utilizada para o acúmulo das mensagens de erro geradas, como por exemplo quando não se obtém sucesso no envio da fila de dados coletados, onde a data e hora do evento e o código de erro são salvos.

Porém, como a maioria dos microcontroladores possuem memória RAM da ordem de *kbytes*, dificilmente se conseguiria passar mais que alguns minutos, talvez horas, a depender da aplicação, desconectado da rede, sem que se haja um *overflow* da fila e consequentemente perda de informações. Portanto, a fim de se evitar a sobrecarga de memória, ao identificar o enchimento da fila, esta é compactada.

Caso o sensor forneça dados acumulativos, como por exemplo em medições de consumo de água e energia, onde os valores coletados a cada intervalo de tempo predeterminado são somados, a compactação é feita perdendo-se resolução temporal. Já no caso de medições temporais, como temperatura e luminosidade, a compactação é feita por meio da exclusão de uma medição a cada duas consecutivas coletadas.

A identificação da quantidade de memória disponível restante é feita através de um algoritmo onde são considerados o tamanho, em bytes, de cada leitura do sensor multiplicado pela quantidade de elementos já armazenados. O resultado do produto é ainda somado ao espaço ocupado pela fila de erros que, por fim, é subtraído da quantidade de memória RAM disponível para uso.

Já na compactação, os elementos da fila tem sua resolução e leitura somada aos pares, reduzindo assim, a memória ocupada pela metade.

2.2 memória FLASH

Diversos sistemas e microcontroladores são equipados com uma memória não volátil, visto que em inúmeras aplicações faz-se necessário que dados sejam recuperados após eventuais quedas de energia ou desenergizações. Geralmente a tecnologia utilizada em tais memórias não voláteis é a EEPROM (electrically erasable programmable read-only memory). Dessa forma, a EEPROM tem como principal objetivo guardar informações para que seja possível acessá-las a qualquer instante. No presente caso, pode-se utilizar a memória não volátil para gravar, por exemplo, o valor acumulado de um contador referente à um sensor. Assim, caso o microcontrolador venha a ser reinicializado, os dados poderão ser recuperados. Porém, o ESP8266 não tem uma EEPROM, e sim uma emulação em um segmento da FLASH de 4 *bytes* até 4M *bytes*, em que cada *byte* da EEPROM emulada possui um limite de dez mil gravações. Caso ultrapasse o valor de gravações permitido a memória irá falhar (Kolban, 2015). Dessa forma, foi desenvolvido uma classe na linguagem C++ para que dispositivos baseados em IoT possam utilizá-la para facilitar a utilização da memória não volátil com o principal objetivo de prolongar a vida útil da EEPROM. Isto é realizado fazendo a gravação na FLASH de forma circular, mudando constantemente o endereço de gravação.

Para gravação circular é necessário reservar endereços da memória para a gravação, de modo que um deles como, por exemplo, o primeiro endereço, é destinado a guardar o último endereço onde foi realizada a última gravação, assim, caso o microcontrolador reinicie, é possível saber em qual endereço o dado foi gravado pela última vez para que seja recuperado. A quantidade de endereços reservados é baseada em quanto tempo deseja-se prolongar a vida útil da memória. A Figura 3 mostra cinco endereços destinados à guardar dados, de modo que a gravação é feita um número de vezes em cada endereço passando para a gravação no seguinte apenas quando completar o número de gravações especificado para a classe, quando termina de gravar o número de vezes especificado no último endereço informado, volta a gravar no endereço inicial.

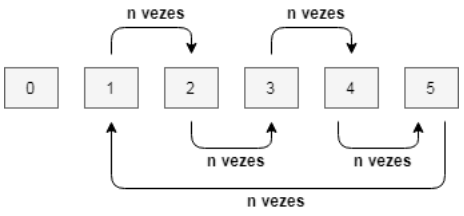

Figura 3: Gravação circular na memória FLASH

Na Figura 4 pode-se observar os métodos e os atributos listados e na tabela 1 a descrição da funcionalidade de cada método presente na classe. Para permitir uma inicialização direta dos objetos da classe, o compilador requer uma função para chamar quando a variável é criada, ou seja, o com-

pilador chama um construtor. Assim, o construtor recebe o endereço que deve ser usado para guardar o endereço onde ocorreu a última gravação, o tamanho do tipo do dado, em *bytes*, que deve ser gravado e a quantidade de posições da memória que vai ser usada para gravar as informações. Determina-se o endereço inicial e final que serão utilizados baseando-se no número de posições e no tamanho do dado.

Como forma de validar o presente trabalho, implementou-se o framework proposto em uma aplicação IoT, que consiste em um sistema automatizado de medição que, por meio de um sistema microcontrolado, coleta os dados de consumo de um hidrômetro, equipamento que mede o volume de água que flui através de uma tubulação, equipado com um sensor digital. Dentro do laboratório de robótica da UFRN (LII), um hidrômetro equipado com a tecnologia descrita nesse artigo foi colocado em um recipiente com uma bomba, que gera um fluxo de água para dentro do mesmo com uma vazão de aproximadamente $100l/h$.

As informações coletadas, são transmitidas via internet a um servidor, onde é feito o armazenamento e análise desses dados, disponibilizando o volume total consumido em uma aplicação *WEB*. A Figura 5 mostra o diagrama de uma aplicação que pode utilizar o framework proposto.

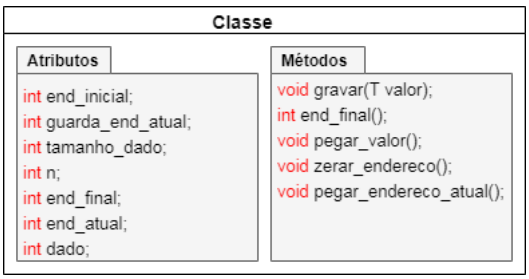

Figura 4: Diagrama da classe de salvar dados na memória FLASH

3 Resultados

Com a implementação das soluções em aplicações reais, percebeu-se ganhos na vida útil da EEMPROM do microcontrolador utilizado como também na persistência dos dados, permitindo que o hardware fique sem conexão com a rede em situações adversas, descartando a necessidade de enviar dados constantemente e possibilitando acompanhar o funcionamento interno do hardware.

3.1 estrutura de dados - fila

No exemplo do leitor de consumo hídrico, abordado no artigo, a memória RAM do dispositivo inicia-se com $37Kbytes$ disponíveis,a qual em cada medição do sensor $56bytes$ são ocupados, além de

Tabela 1: Descrição dos métodos da classe que grava informações na memória FLASH do ESP8266.

Métodos	**Descrição**
zerar_endereco();	Especifica o endereço que deve ser usado para guardar em qual endereço ocorreu a última gravação, bem como o endereço onde começará a gravação e zera o conteúdo do endereço.
pegar_end_atual();	Pega o endereço onde ocorreu a última gravação.
pegar_valor();	Pega o valor que está no endereço onde ocorreu a última gravação e coloca em uma variável.
gravar(T dado);	Grava uma informação em um endereço da memória, o dado pode ser de qualquer tipo. Porém, é necessário mandar o seu tamanho no construtor da classe.

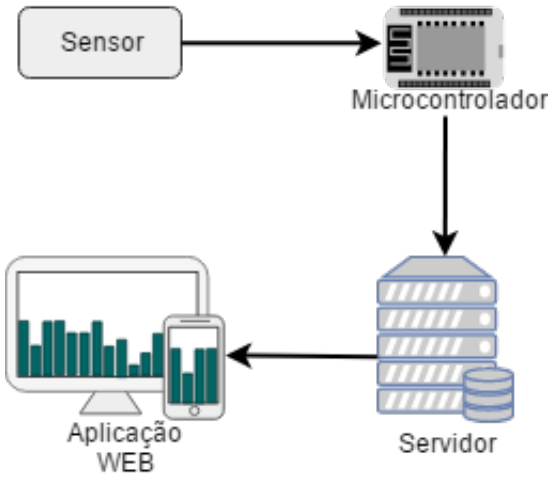

Figura 5: Esquema da aplicação utilizada para teste do framework proposto

cada nova inserção na fila de erros ocupa, no máximo $360bytes$.

Durante os experimentos, a fila obteve o comportamento descrito na Figura 6, sendo possível observar uma capacidade de armazenamento de cerca de 250 elementos sem que seja necessário recorrer à compactação. A partir dessa observação, estima-se ser possível suportar até $41,7$ horas, sem a necessidade de se recorrer a compactação, sendo os dados enviados do dispositivo ao servidor

a cada meia hora.

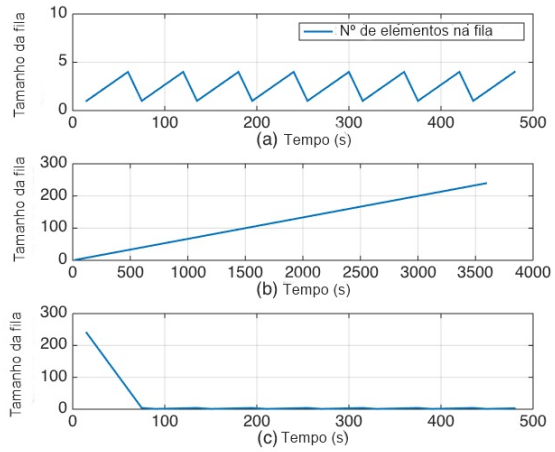

Figura 6: Comportamento da fila durante os experimentos. Em (a) pode-se perceber o comportamento normal da estrutura quando há internet disponível. O tamanho da fila cresce enquanto são inseridos novos elementos e decresce quando estes são enviados ao servidor. Em (b), quando a conexão com a internet é perdida, o tamanho da fila aumenta até que a conexão seja reestabelecida. Em (c) a conexão foi reestabelecida e o dispositivo voltou a enviar os dados salvos, esvaziando a fila e retornando ao seu comportamento normal como descrito em (a).

3.2 gravação na memória FLASH do ESP8266

A utilização da classe que grava dados na memória FLASH é indicada para aplicações que possuem sensores do tipo contador, como é o caso da aplicação que contabiliza a quantidade de litros de água que passa por uma tubulação, nessa situação o valor da variável que contém a quantidade de litros já passados pelo medidor de água não pode ser perdida caso o microcontrolador reinicie, pois os dados precisam ser enviados ao Servidor e mostrados na aplicação Web. Dessa forma, a classe foi implementada para a medição automática do consumo de água, em que utilizou-se dez endereços de memória e tendo que gravar dez vezes em um endereço para poder gravar no próximo. Para gravação, a utilização de uma interrupção foi vista como a melhor maneira, visto que com esse recurso pode-se chamar a função de gravar de forma prioritária. Esse mecanismo interrompe o processamento atual do microcontrolador e realiza uma tarefa especificada (chamada interrupt service routine ou ISR); uma vez concluída a ISR, o processador retoma a execução do ponto onde foi interrompido. A figura 7 mostra como a classe foi implementada no *firmware* da aplicação que contabiliza a quantidade de litros que passa pelo hidrômetro. Para essa aplicação achou-se necessário gravar o valor do contador de litros a cada dez minutos na memória FLASH. Sem a gravação circular a vida útil da memória seria de aproximadamente 70 dias, já com a gravação circular e adotando as condições para essa aplicação a vida útil da memória se prolonga para 695 dias, o que representa um aumento de cerca de 1000% .

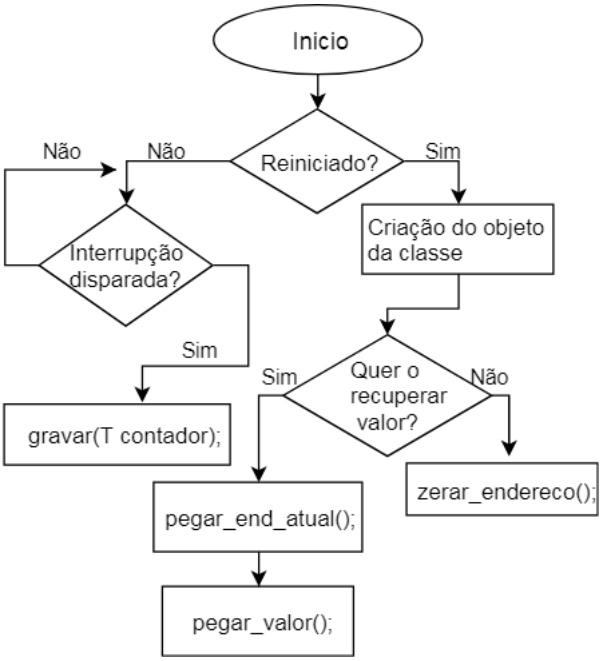

Figura 7: Diagrama da classe de salvar dados na memória FLASH

4 Conclusões

O presente trabalho apresentou um conjunto de soluções para o desenvolvimento de dispositivos de hardware robustos e de baixo consumo baseados na arquitetura de IoT.

Considerando as diversas adversidades a que estes dispositivos estão sujeitos, como a perda momentânea da conexão de rede, foi proposta a implementação de uma estrutura de dados do tipo fila, na qual os dados provenientes dos sensores são acumulados, até que a conexão seja reestabelecida, quando enfim podem ser enviados ao servidor, ou mesmo com o propósito de economia de energia, sendo acumulados por intervalos pre-definidos de tempo antes de serem enviados em lote.

Em caso de perda de alimentação, a perda de informações ainda não enviadas, é evitada através da gravação periódica na memória FLASH do microcontrolador. Para isso, foi abordado o desenvolvimento de uma classe que faz a gravação de dados de forma circular que tem por consequência o prolongamento da vida útil da memória.

A proposta foi validada em dispositivos reais de medição de consumo de água e tem se mostrado eficaz, visto que não perdeu dados com a falta de energia elétrica ou reinicio do dispositivo

e houve o prolongamento da vida útil da memória não volátil do microcontrolador. Quanto ao uso da estrutura de dados fila, esta atendeu a necessidade de manutenção dos dados em caso de falta de rede mostrando-se capaz de suportar, sem recorrer a compactação, um tempo considerável para que possíveis problemas que impeçam a conexão sejam resolvidos e o dispositivo possa voltar ao seu comportamento normal.

Como trabalho futuro destacamos a realização de testes com dispositivos de tipos distintos, como por exemplo os de medições temporais.

Agradecimentos

Agradecemos a Universidade Federal do Rio Grande do Norte (UFRN) pelo apoio ao desenvolvimento deste trabalho, ao Núcleo de Pesquisa e Inovação em Tecnologia da Informação (NPITI) pelo espaço físico disponibilizado, também aos colegas de laboratório pelo suporte e apoio.

Referências

Ahmed, E. and Rehmani, M. H. (2017). Mobile edge computing: Opportunities, solutions, and challenges, *Future Generation Computer Systems* **70**: 59 – 63.

Albino, V., Berardi, U. and Dangelico, R. M. (2015). Smart cities: Definitions, dimensions, performance, and initiatives., *Journal of Urban Technology* **22**(1): 3 – 21.

Azarov, V. N. and Maiboroda, V. P. (2017). Synergetic of information and communication technologies and quality tools in the tasks of engineering management, *2017 International Conference "Quality Management,Transport and Information Security, Information Technologies"(IT QM IS)*, pp. 463–465.

Espressif (2018). Esp8266. https://www.espressif.com/en/products/hardware/esp8266ex/overview.

Kolban, N. (2015). Kolban's book on esp8266, *an introductory book on ESP8266* .

McRoberts, M. (2011). Arduino básico, *São Paulo: Novatec* .

Satyanarayanan, M. (2017). Edge computing, *Computer* **50**(10): 36–38.

Soliman, M., Abiodun, T., Hamouda, T., Zhou, J. and Lung, C.-H. (2013). Smart home: Integrating internet of things with web services and cloud computing, *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, Vol. 2, IEEE, pp. 317–320.

WEMOS Electronics (2018). D1 mini. https://wiki.wemos.cc/products:d1:d1_mini.

Zhang, D., Ning, H., Xu, K. S., Lin, F. and Yang, L. T. (2012). Internet of things, *Int. J. Communication Systems* **25**: 1101–1102.

Zhuhadar, L., Thrasher, E., Marklin, S. and de Pablos, P. O. (2017). The next wave of innovation—review of smart cities intelligent operation systems, *Computers in Human Behavior* **66**: 273 – 281.