Automação Predial Inteligente com a Nuvem – Sensores e Atuadores na Nuvem: RTOS

Li Wing Kee Ramos*, **Adolfo Bauchspiess** *

* *Laboratório de Automação e Robótica - LARA, UnB, Brasília, DF,*
e-mails: liwramos1@gmail.com, adolfobs@ene.unb.br.

Abstract: Internet of Things is the medium in which billions of devices are connected to the Internet transmitting and receiving data in order to make all environments and processes more intelligent. These *things* can be defined as embedded devices, based on microcontrollers, that connect to the network. These devices are limited in memory size and power consumption, however, for the context of building automation, IoT devices with increasingly connectivity, security, and reliability are indispensable, so following all these pre-requisites becomes a challenging task. Thus, Real Time Operating Systems (*RTOS*) emerge as a resource management tool for these microcontrollers, which adds a series of functionalities in order to facilitate project development and ensure robustness to the system.

Resumo: Internet das Coisas é o meio no qual bilhões de dispositivos estão conectados à internet transmitindo e recebendo dados de forma a tornar todos os ambientes e processos mais inteligentes. Essas *coisas* podem ser definidas como dispositivos embarcados, baseados em microcontroladores, que se conectam à rede. Esses dispositivos são limitados em tamanho de memória e consumo de energia, no entanto, para o contexto de automação predial, é imprescindível dispositivos IoT com cada vez mais conectividade, segurança, e confiabilidade, assim, atender a todos esses requisitos se torna uma tarefa desafiadora. Neste contexto, os Sistemas Operacionais de Tempo Real (*RTOS*) surgem como uma ferramenta de gerenciamento de recursos para os microcontroladores utilizados no *motes* IoT, que adicionam uma séries de funcionalidades de forma a facilitar o desenvolvimento do projeto e garantir robustez ao sistema.

Keywords: Building automation, Real-time Operating Systems, LoRa, IoT.
Palavras-chaves: Automação predial, Sistemas Operacionais de Tempo Real, LoRa, IoT.

1. INTRODUÇÃO

A arquitetura de um sistema IoT envolve uma grande quantidade de nós-terminais, ou dispositivos de borda (geralmente microcontroladores), conectados a dispositivos agregadores (gateways), que por sua vez se conectam à internet, integrando todo o sistema à nuvem.

Em aplicações simples, desenvolvedores de software para microcontroladores, implementam suas aplicações baseando-se na na metodologia de *superloop*, em que o programa principal é executado sequencialmente dentro de um laço infinito. Essa abordagem é adequada enquanto o dispositivo executa um número limitado de funções. Segundo Sabri et al. (2018), com o aumento do número de periféricos e protocolos de comunicação, se faz necessário o uso de ferramentas que facilitem a integração desses elementos.

É crescente o número de aplicações de microcontroladores baseados em Sistemas Operacionais de Tempo Real, ou RTOS, da sigla em inglês. Para Lethaby (2013) a principal motivação para a implementação de um RTOS em um projeto de IoT é a complexidade da aplicação. Frequentemente, esses Sistemas Operacionais são utilizados quando existem muitas fontes de interrupção, como sensores e telas de interação com usuário, ou quando existem muitas interfaces de comunicação no projeto, como Wi-fi e Bluetooth.

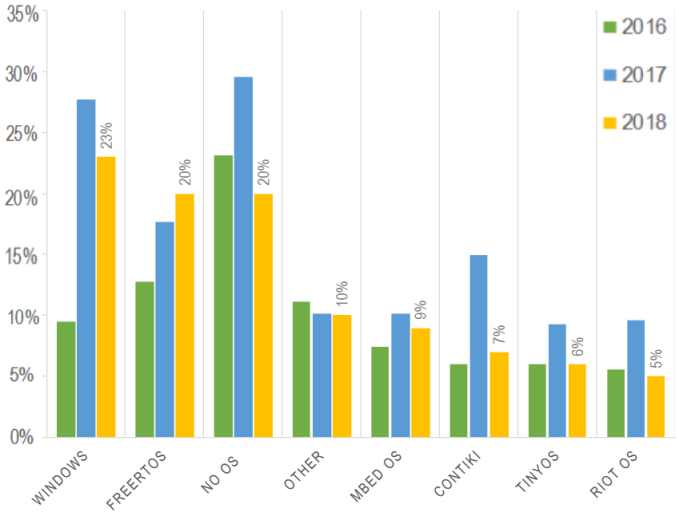


Figura 1. *IoT Survey 2018* - Uso de SO em projetos IoT.

De acordo com Skerett (2018), sistemas operacionais *Linux-like* dominam o mercado de IoT, sendo adotados em de 71% dos projetos no ano de 2018, tanto para dispositivos de borda quanto para gateways. Dentre os sistemas operacionais de tempo real, o FreeRTOS foi o mais utilizado, sua adoção foi de 14% em 2016 para 20%

em 2018. Os principais fatores para o crescimento do uso desse sistema é a ampla compatibilidade entre diferentes arquiteturas e a licença gratuita para comercialização.

Para Brad e Murar (2014), os principais desafios para a implementação de sistemas IoT para ambientes inteligentes são os riscos de segurança, custo de desenvolvimento do sistema, flexibilidade de construção do produto, e o grande número de dispositivos conectados simultaneamente. Esses desafios levantam uma série de pré-requisitos que um RTOS deve seguir, como escalabilidade, modularidade, conectividade, e segurança.

Modularidade significa que as partes do projeto são separáveis, assim, é mais simples adicionar novas funcionalidades, e substituir tecnologias. Escalabilidade é necessário para adaptar o sistema para uma arquitetura com cada vez mais nós, e processos mais complexos. Conectividade implica que a rede deve ser compatível com diversas interfaces de comunicação, e ao mesmo tempo a segurança deve ser robusta de modo a garantir o correto funcionamento do sistema e impedir ataques maliciosos que possam causar danos a vidas, ou mesmo econômicos.

2. FUNDAMENTAÇÃO TEÓRICA

Segundo M. Hussein (2016), a fim de se desenvolver o sistema de automação para um ambiente predial, é imprescindível a precisão do controlador e a robustez da comunicação dos sensores entre si, e com a nuvem, pois é a partir da integração desses dispositivos que garante-se o controle eficiente da planta. É nesse contexto que os sistemas operacionais de tempo real surgem como uma solução ideal para o projeto.

2.1 FreeRTOS

FreeRTOS é um *kernel* de tempo real (ou escalonador de tempo real) usado para o desenvolvimento de sistemas embarcados. Ele permite subdividir a aplicação em tarefas, que podem ser executadas em um ou em vários núcleos independentemente. O papel do escalonador é decidir qual tarefa deve ser executada pelo processador, a escolha é feita com base nas prioridades que são designadas para cada tarefa assim que ela é declarada.

As tarefas são executadas ciclicamente, e o período de repetição é dado pelo parâmetro *TaskDelay()*, quando a tarefa chega ao final da execução, ela é movida para o estado de bloqueada, e assim que o tempo de delay termina, ela é movida para o estado de pronta, esse tipo de atribuição permite que o processador não tenha que fazer *pooling* para aguardar algum evento, ou dependa de flags de interrupção que podem prejudicar a função principal.

Conforme Silva et al. (2019), a troca de tarefas é feita de forma preemptiva ou cooperativa. No modo preemptivo, a tarefa que está sendo executada pode ser interrompida arbitrariamente caso uma tarefa de maior prioridade entre no estado de pronta, já no modo cooperativo, é necessário que a tarefa que está sendo executada entre no estado bloqueado ou execute um comando que permite outra tarefa ser executada.

Como as tarefas ocorrem de forma assíncrona, a interação entre esses contextos deve ser sincronizada, de tal modo

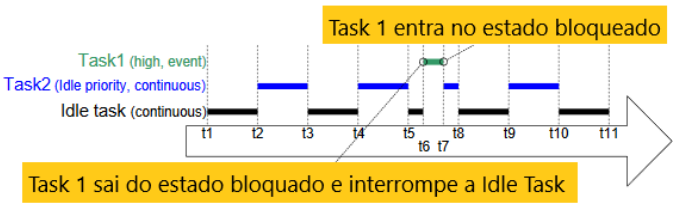


Figura 2. Escalonador preemptivo. Barry (2018)

que essas tarefas não acessem o mesmo recurso compartilhado ao mesmo tempo. A principal ferramenta para essa função são os semáforos binários, sua implementação consiste em criar uma estrutura *Mutex* (Mutuamente exclusiva), que só pode ser alocada para uma tarefa por vez. Assim, quando uma tarefa precisa utilizar de um recurso compartilhado, utiliza-se a função *xSemaphoreTake(Mutex)*, e assim que a execução terminar, devolve-se o mutex através de *xSemaphoreGive(Mutex)*. Dessa forma, o recurso fica protegido enquanto o mutex estiver alocado a uma tarefa.

2.2 LoRa

De acordo com Sanchez-Iborra et al. (2018), a rede LoRa é uma tecnologia de comunicação sem fio desenvolvida para aplicações que demandam baixa potência e longo alcance, por isso se enquadra como uma rede LPWAN(*Low Power Wide Area Network*), a maior limitação desse tipo de conexão é a baixa taxa de transmissão de dados, se comparado com outras redes de menor alcance. Mas a confiabilidade, e a eficiência da comunicação tornam essa rede ideal para aplicações IoT.

É de se notar que essa tecnologia apenas especifica a técnica de modulação do sinal. O protocolo distribuído oficialmente pela *LoRa Alliance* é o *LoRaWAN*, que estabelece uma topologia em estrela no qual os gateways retransmitem os dados entre os dispositivos de borda. Essa tecnologia é aberta, e permite a implementação de protocolos customizáveis e que se adéquem melhor às especificações de cada projeto.

2.3 openHAB

O openHAB, acrônimo de *open Home Automation Bus*, é um software livre de automação que visa a integração de diferentes sistemas e tecnologias em uma única solução, pode ser controlado por uma interface gráfica acessível de qualquer navegador padrão ou aplicativos de celular.

Este software serve como central de integração que se coloca entre os hardwares dos dispositivos. Dessa forma, apenas a instância do openHAB necessita estar rodando em um servidor central, e também possui um SO próprio, o *openhabian*, que é uma versão de Linux já com todos os pacotes necessários instalados, de forma que facilita a instalação e desenvolvimento, principalmente em hardwares mais limitados como o Raspberry Pi.

Não é necessário conexão com a nuvem, mas possui amplo suporte a essa tecnologia, e inclusive possui um servido próprio chamado *myopenhab*. Dessa forma, o openHAB se torna uma ferramenta completa de monitoramento, controle, e integração com a nuvem, para sistemas de automação predial.

2.4 MQTT

O MQTT, sigla de *Message Queuing Telemetry Transport*, é um protocolo de comunicação assíncrono M2M (*machine-to-machine*), que é otimizado para operar sobre o protocolo TCP/IP. Por isso, para Piper (2013) é um protocolo ideal para aplicações IoT, visto que pode ser implementado em dispositivos com pouca memória e com poder de processamento limitado.

O protocolo MQTT segue o modelo *publish/subscribe* (publicar/subscrever (assinante de um serviço)), assim, existem duas entidades na rede, o *broker* e o *client*. O papel do *broker* é ser um servidor que recebe, e organiza os dados recebidos em tópicos, e os direcionam para os *clients* que se inscreveram (assinaram) no tópico. O *client* é qualquer dispositivo que pode se conectar ao *broker*, para mandar ou receber mensagens.

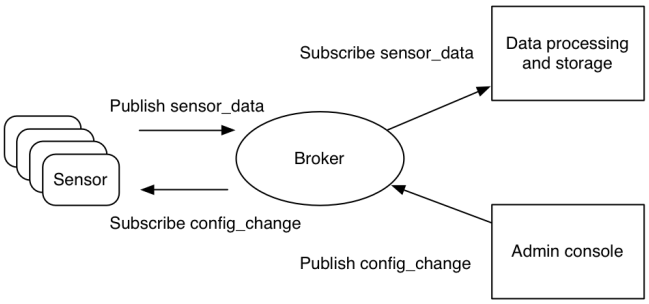


Figura 3. Esquemático MQTT,Piper (2013).

3. METODOLOGIA

O objetivo do sistema de automação foi monitorar e controlar a temperatura de um ambiente fechado. A figura 4 apresenta um exemplo de disposição dos sensores em um ambiente similar ao LARA (Laboratório de Automação e Robótica). Onde a sala de reuniões é o ambiente no qual foi feito o controle de temperatura.

3.1 Visão Geral do Sistema de Automação

Ao todo, utilizam-se três sensores de temperatura e umidade DHT22. O primeiro é o disposto dentro do ambiente controlado. O segundo fica no porta de entrada e saída, que é o principal ponto de troca de calor. E o último sensor é alocado para o ambiente externo, no qual é possível medir informações a respeito do tempo.

Cada sensor *DHT22* está conectado a um microcontrolador ESP32, que é responsável por processar, e transmitir os dados obtidos de cada ambiente. Um quarto ESP32 foi instalado no centro da sala de reuniões, e este irá operar como um gateway, que envia todos os dados recebidos para o servidor local, e transmite comandos recebidos para os nós. Dessa forma, estabelece-se uma topologia em estrela, em que os nós comunicam-se apenas com um dispositivo central, e não entre si (diretamente).

O nó central da rede, o gateway, é o único que estabelece conexão direta com o servidor local, este, por sua vez, roda em um *Raspberry Pi* instalado com *openHABian*, assim,

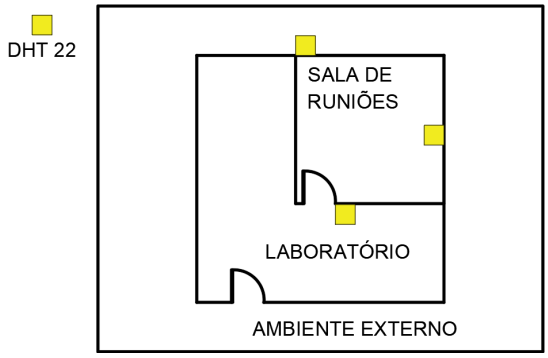


Figura 4. Disposição dos sensores na maquete caseira. Construção de Li W.K. Ramos (Contingência COVID-19).

é possível simplificar a comunicação entre os dispositivos através do esquemático da figura 5.

Obs.: Na maquete todos os sensores poderiam, facilmente, ser conectados no mesmo ESP32, no entanto, o objetivo aqui é emular a automação de um prédio real.

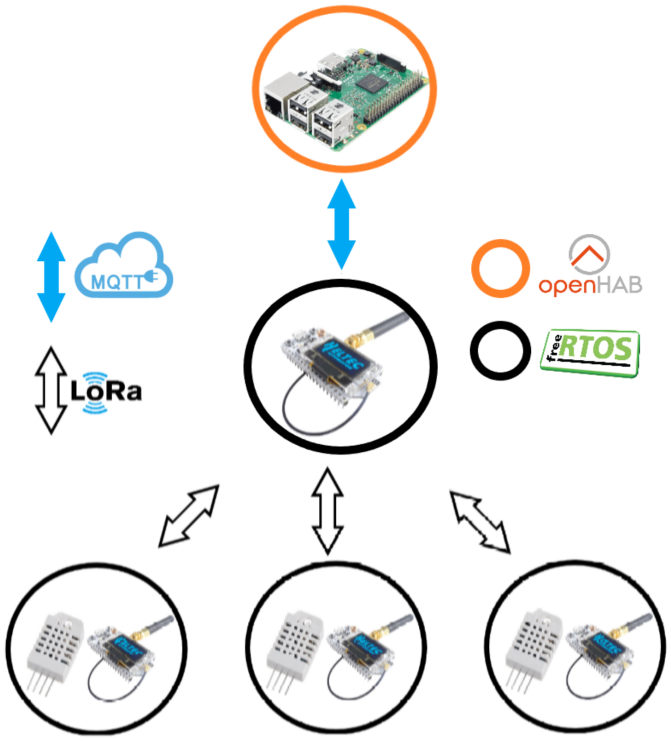


Figura 5. Comunicação entre os dispositivos.

3.2 ESP32

É um microcontrolador de baixo custo desenvolvido pela empresa *Espressif*, (*espressif.com*). A principal vantagem dessa série de chips é sua conectividade. Possui suporte nativo à interface Bluetooth, WiFi, e existem versões de outros desenvolvedores, como o caso da *Heltec*, que distribuem versões com rádio *LoRa* também embutido. Além disso, é equipado com dois núcleos físicos de processamento *Xtensa LX6*, o que permite multiprocessamento paralelo,

.

A plataforma de desenvolvimento optada para a programação dos microcontroladores foi o *ESP-IDF (Espressif*

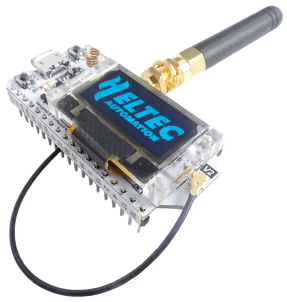


Figura 6. ESP32 - Heltec LoRa.

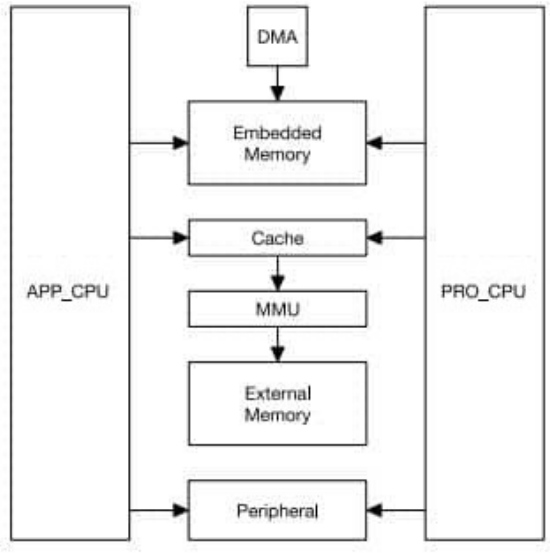


Figura 7. Arquitetura ESP32.Vagapov (2017)

IoT Development Framework), que foi desenvolvida pela própria fabricante dos dispositivos. Esse ambiente de desenvolvimento oferece suporte nativo ao *freeRTOS* e possui ferramentas e recursos que garantem desempenho e robustez a projetos de IoT.

3.3 Dispositivos de Borda

Os dispositivos de borda são os nós da rede responsáveis pela atuação e sensoriamento do sistema, todos esses nós se comunicam sem fio, através da rede LoRa. Sendo assim, utilizam os dois núcleos do processador do ESP32 de forma paralela. É delegado ao *app_cpu* a tarefa de receber os valores de temperatura e umidade medidos pelo sensor, e analisar os dados de forma a realizar o controle de temperatura do ambiente, enquanto o *pro_cpu* é responsável pela comunicação LoRa, tanto receber quanto enviar mensagens.

É dado o nome de *Control_Task* à tarefa de medir a temperatura, e realizar o controle da planta. Implementou-se, inicialmente, o controlador Liga-Desliga (com intervalo diferencial, $\lambda_1, \lambda 2$). O erro é dado pela diferença entre referência e sinal e temperatura medida:

$$e = r - y,$$

$$u(t) = 220V, \text{ se } e > \lambda_1,$$

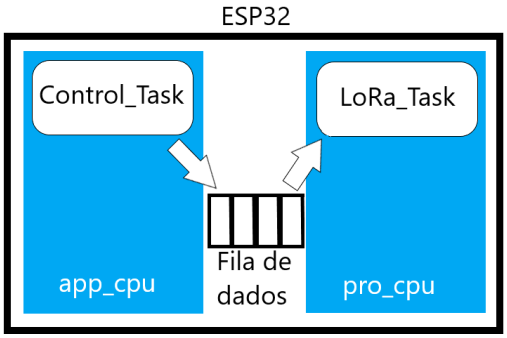


Figura 8. Divisão de tarefas nos dispositivos da borda.

$$u(t) = 0V, \text{ se } e < \lambda_2.$$

A figura 9 ilustra o fluxograma do algoritmo implementado na tarefa *Control_Task*. Nota-se que, a fim de se evitar redundância dos dados enviados pelos sensores, foi adicionado a condição de que o valor medido atual deve ser diferente da medição anterior para ser enviado para a fila de dados da rede LoRa.

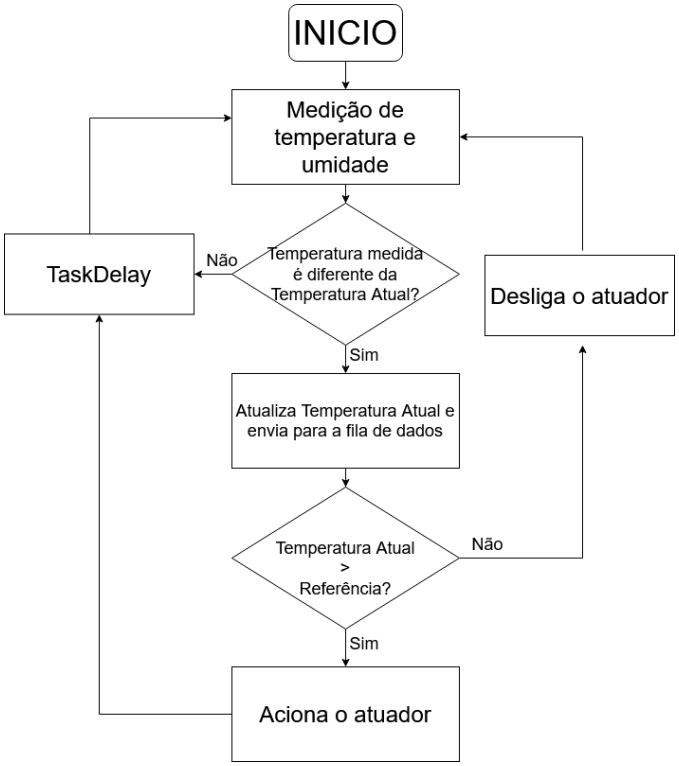


Figura 9. Algoritmo *Control_Task*

Paralelamente à *Control_Task*, é realizado a comunicação LoRa dos microcontroladores. Nesse tipo de tecnologia, o dispositivo é configurado de forma que pode apenas transmitir, ou receber mensagens, não é possível fazer os dois simultaneamente, sendo assim, é importante que por padrão o ESP32 esteja no modo de receber dados, e apenas se tiver alguma atualização de temperatura na fila de dados ele deve alternar para o modo de transmissão.

A fim de aumentar a confiabilidade de que a informação chegou ao gateway, implementou-se também um sistema de *Acknowledgement*, isto é, assim que o gateway recebe uma informação, checa-se a integridade dos dados através de checksum, se a informação for validada, é enviado ao

nó um sinal de *ACK*, confirmando que o dado foi recebido. Caso o nó não receba essa confirmação, é feito uma nova tentativa de reenviar o mesmo dado.

A tarefa responsável por interfacear a comunicação LoRa, através do protocolo criado, é chamada de *LoRa_Task*, e seu algoritmo implementado pode ser visualizado no fluxograma da figura 10. Todos os ACKs são identificados com timestamp.

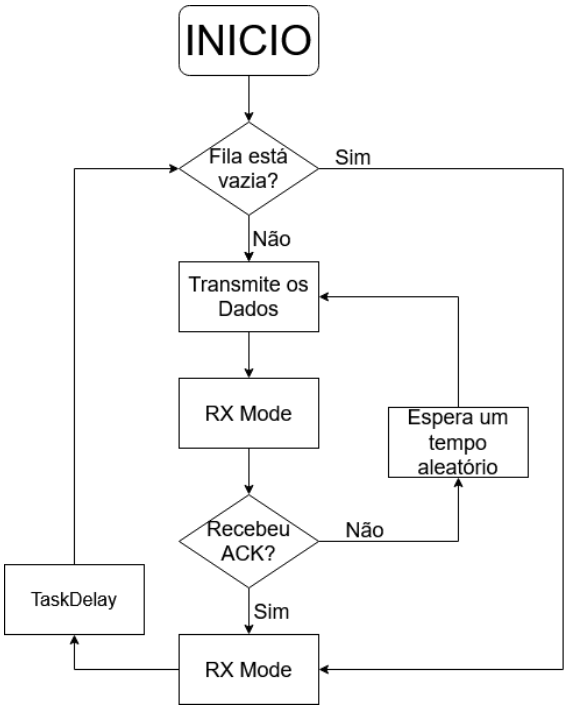


Figura 10. Algoritmo *LoRa_Task*

3.4 Gateway

O papel desse tipo de dispositivo é servir como uma ponte de ligação entre nós da rede que se comunicam por protocolos de comunicação diferentes. Dessa forma, de acordo com Chandra (2016), os gateways habilitam a comunicação entre diferentes arquiteturas e ambientes. Assim, no contexto de internet das coisas, esses dispositivos são os responsáveis por estabelecer a conexão entre uma rede local e a nuvem.

Aplicando-se esse conceito à rede *LoRa* é possível combinar as principais vantagens dessa tecnologia, como a robustez e o longo alcance sem precisar abrir mão da conexão com a internet, que muitas vezes é fundamental para o controle do processo. Assim, essa solução se torna adequada para uma situação de automação predial, em que existe demanda para a integração de todo o sistema.

De forma similar aos dispositivos de borda, os gateways também utilizam o sistema operacional *freeRTOS*, e cada um dos processadores roda um conjunto de tarefas paralelamente. A recepção de mensagens *LoRa* é realizada pelo *app_cpu*, enquanto o *pro_cpu* realiza o interfaceamento com o *broker MQTT*, como ilustra a figura 11.

O papel da tarefa *MQTT_Task* é de receber dados provenientes da nuvem e deposita-los na fila de dados para que *LoRa_Task* transmita o dado para o nó correto. O caminho

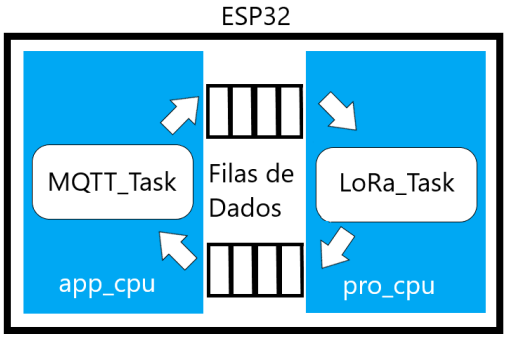


Figura 11. Divisão de tarefas do gateway.

contrário também ocorre, ou seja, os nós terminais enviam dados através da rede LoRa, e os dados são transmitidos para a nuvem via MQTT.

3.5 Ambiente de testes

Para a realização do experimento de controle de temperatura, visto que no local utilizado não existe sistema de ar-condicionado, utilizou-se uma caixa de papelão (dimensões aproximadas, $33x47x27\ cm^3$) com o intuito de simular um ambiente climatizado, assim como foi utilizado em Santos (2017), e ao invés de utilizar do ar-condicionado para resfriar o ambiente de modo a manter a temperatura constante, utilizou-se um secador de cabelo para aquecer a caixa, dessa forma, basta inverter a desigualdade da tarefa da figura 9 para que o experimento seja adaptado a essa nova condição.

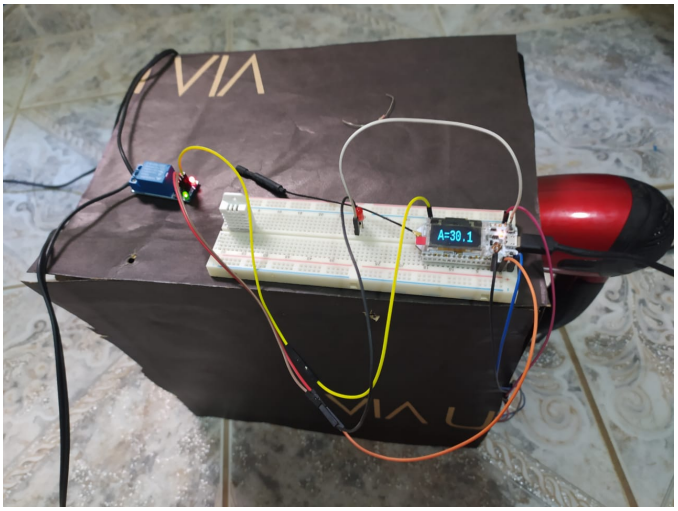


Figura 12. Simulação - Ambiente fechado.

3.6 Teste RTOS x SuperLoop

A fim de se mensurar o ganho de performance com a utilização de sistemas operacionais em tempo real, foi feita a comparação da implementação do sistema com o FreeRTOS, e sem nenhum sistema operacional, com a metodologia de *superloop*, que consistiu em implementar o mesmo código das tarefas dos controladores de forma sequencial em um único laço infinito. Em seguida, verificou-se qual dos códigos é executado mais vezes por minuto.

O teste de estresse consistiu em submeter o controlador a uma condição atípica de operação, no qual o gateway

envia informações via LoRa a cada 10 ms, enquanto a temperatura do ambiente é controlada. O *TaskDelay* de cada tarefa foi configurada para ser a menor possível, de forma que o processador esteja sempre consumindo linhas de código. Utilizou-se do Timer interno do microcontrolador para gerar uma interrupção a cada minuto que imprime no terminal a quantidade de vezes que determinada tarefa foi executada naquele intervalo de tempo.

4. RESULTADOS

Na tabela 1 verifica-se que a implementação do código em *superloop* executou menos vezes por minuto do que a aplicação do FreeRTOS utilizando apenas um núcleo. A presença do escalonador permite que a execução do programa não tenha que ficar aguardando a medição de temperatura para realizar a transmissão de dados. Assim, a comunicação do microcontrolador não depende do sensor e nem do tipo de controlador implementado.

Tabela 1. Teste sob condições normais.

	Execuções por minuto		
Tarefas	**FreeRTOS (dual-core)**	**FreeRTOS (single-core)**	**Sem RTOS**
Medição/ Controle	138	111	92
Comunicação LoRa	750	509	92

Submetendo-se o microcontrolador a condição de esstresse, isto é, reduzindo-se ao máximo o delay entre as tarefas, e enviando mensagens pelo gateway em uma frequência elevada, fica ainda mais evidente o ganho de desempenho do processador com a utilização do FreeRTOS.

Tabela 2. Teste sob stress.

	Execuções por minuto		
Tarefas	**FreeRTOS (dual-core)**	**FreeRTOS (single-core)**	**Sem RTOS**
Medição/ Controle	1104	827	462
Comunicação LoRa	3642	2320	462

Em um sistema de controle digital, normalmente, a taxa de amostragem é constante. Como o processo térmico é lento, 1 s ou até mesmo 1 min, seriam suficientes. As tabelas mostram, assim, que com FreeRTOS dual core seria possível, executar 1104 vezes por minuto. Desse modo, algoritmos de controle mais sofisticados, como "controle preditivo"podem ser executados por um ESP3 em tarefas de automação predial.

Observa-se, na figura 13, o resultado do controle da temperatura da caixa, em que o secador foi acionado 2 minutos após o início do experimento. A temperatura inicial foi de 35 °C e a referência 50 °C. É de se notar que a temperatura ambiente seguiu bem a referência, com um erro aproximado de ±0.2 °C. Após 34 minutos, a temperatura de referência foi reduzida para 35 °C novamente.

Já na figura 14, tem-se os histórico de medição de temperatura e umidade ao longo de 12 horas, sem nenhum tipo de controle sobre o ambiente.

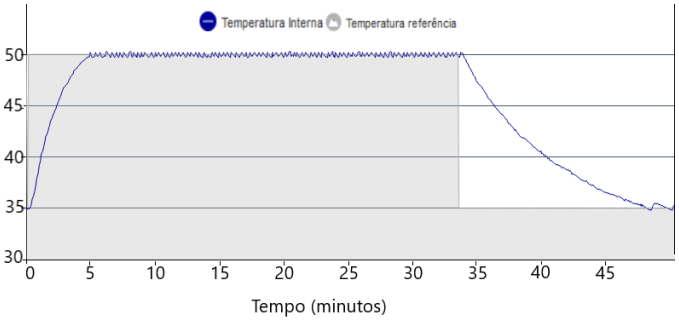
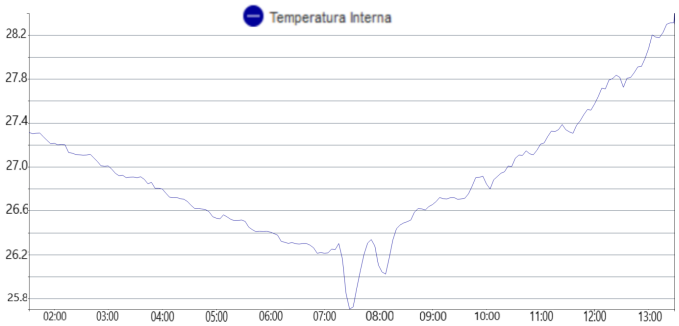
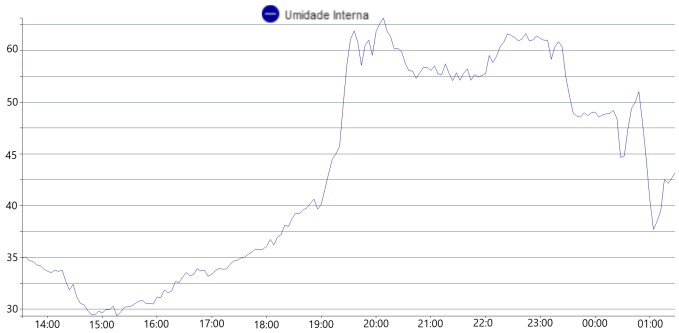


Figura 13. Temperatura - controle liga-desliga



(a) Temperatura do Quarto.



(b) Umidade do Quarto

Figura 14. Histórico de temperatura e Umidade no quarto (medido em 14/10/2020)

5. CONCLUSÃO

À primeira vista, sistemas operacionais de tempo real podem aparentar aumentar o nível de complexidade de um projeto de IoT, e portanto, dificultar a implementação de sistemas de controle. No entanto, o que se observou durante a pesquisa, foi que o FreeRTOS facilitou o desenvolvimento do projeto, na medida que possibilitou modularizar os código em diversas tarefas que ocorrem de forma paralela.

É intrínseco, em projetos de sistemas embarcados, a necessidade de esperar algum evento, seja uma *flag* de interrupção, ou a resposta de algum sensor, para dar prosseguimento à execução do código. Tipicamente, em um programa sem escalonador, o processador faz *pooling* até o evento acontecer, por outro lado, como o RTOS opera de forma preemptiva, é possível fazer uso mais eficiente do processador, alocando outras tarefas para serem realizadas enquanto o evento não ocorre.

Outra grande vantagem desse projeto, é a portabilidade do código. A variedade de microcontroladores disponíveis no mercado aumenta todos os anos, e para aplicações de baixo nível, é trabalhoso adaptar essas aplicações para novas arquiteturas. Com o uso do FreeRTOS é mais simples reutilizar o mesmo código, tendo em vista que as *API* são as mesmas independente do dispositivo.

Com o crescente número de sensores, protocolos de comunicação, e periféricos como um todo, se torna cada vez mais complexo e custoso desenvolver aplicações que integrem todos esses dispositivos e tecnologias diferentes.

Nesse contexto, sistemas operacionais de tempo real, se tornam uma excelente alternativa à programação típica de microcontroladores, pois provêm ferramentas que possibilitam o uso muito mais eficiente dos recursos, além de facilitar a integração do dispositivo com a rede.

REFERÊNCIAS

Barry, R. (2018). Mastering the FreeRTOS Real Time Kernel. *Real Time Engineers Ltd*, 2.

Bauchspiess, A, Rodrigues, A, e A.S., da Silva, M.P.M. (2019). Controle Antecipativo por Estimativa de Carga Térmica em Vídeo. *SBA2019, Ouro Preto*.

Brad, B. e Murar, M. (2014). Smart Buildings Using IoT Technologies. *Stroitel'stvo Unikal'nyh Zdanij i Sooruzenij; Construction of Unique Buildings and Structures*, 5(5), 15.

Chandra, T.B. (2016). Operating Systems for Internet of Things. *Proceedings of the Second International Conference on Information and Communication Technology for Competitive Strategies*.

Graham, B.B. e Weinstein, M. (2014). The RTOS as the engine powering the internet of things. *Wind River - White Paper*, 1–3.

Javed, F., Afzal, M.K., Sharif, M., e Kim, B.S. (2018). Internet of Things (IoT) operating systems support, networking technologies, applications, and challenges: A comparative review. *IEEE Communications Surveys and Tutorials*, 2062.

Lethaby, Nick, T.I. (2013). Why Use a Real-Time Operating System in MCU Applications. *White Paper and Articles - Texas Instruments*, 3.

M. Hussein, M. Zorkany, N.S.A.K. (2016). Real Time Operating Systems for the Internet of Things, Vision, Architecture and Research Directions. *Proceedings - 2016 World Symposium on Computer Applications and Research, WSCAR 2016*, 77.

Piper, A. (2013). Choosing Your Messaging Protocol: AMQP, MQTT, or STOMP. *VMware Blogs*, 2.

Sabri, C., Kriaa, L., e Azzouz, S.L. (2018). Comparison of IoT constrained devices operating systems: A survey. *Proceedings of IEEE/ACS International Conference on Computer Systems and Applications, AICCSA*, 2017-October, 369.

Sanchez-Iborra, R., Sanchez-Gomez, J., Ballesta-Viñas, J., Cano, M.D., e Skarmeta, A.F. (2018). Performance Evaluation of LoRa Considering Scenario Conditions. *Sensors*, 5.

Santos, O.L.D. (2017). Sistema de Controle de Temperatura para uma Estufa com Monitoramento via Aplicativo. *Revista de Ciência e Tecnologia - UFRR*, 7.

Silva, M., Cerdeira, D., Pinto, S., e Gomes, T. (2019). Operating Systems for Internet of Things Low-End Devices: Analysis and Benchmarking. *IEEE Internet of Things Journal*, 6(6), 10375.

Skerett, I. (2018). IoT Developer Survey. *Linkedin Slide Share*, 4(April), 18.

Tedesco, L.P. e Oliveira, T.N. (2009). Controle Automatizado De Condicionadores De Ar Utilizando Sistemas Embarcados. *Anais do Salão de Ensino e de Extensão*, 36.

Vagapov, Y. (2017). Comparative Analysis and Practical Implementation of the ESP32 Microcontroller Module for the Internet of Things. *7th International Conference on Internet Technologies and Applications*, 2.