UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Pedro Alexandre Barradas da Côrte

Análise do consumo de energia para dispositivo IoT no monitoramento dos recursos hídricos de larga escala

Florianópolis
2023

Pedro Alexandre Barradas da Côrte

Análise do consumo de energia para dispositivo IoT no monitoramento dos recursos hídricos de larga escala

O presente trabalho em nível de mestrado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

_____

Prof. Alex Sandro Roschildt Pinto, Dr.
Universidade Federal de Santa Catarina

_____

Prof. Roberto Willrich, Dr.
Universidade Federal de Santa Catarina

_____

Prof[a]. Janine Kniess, Dra.
Universidade do Estado de Santa Catarina

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de Mestre em Ciência da Computação.

Documento assinado digitalmente
PATRICIA DELLA MEA PLENTZ
Data: 28/08/2023 13:45:38-0300
CPF: ***.243.700-**
Verifique as assinaturas em https://v.ufsc.br

_____

Prof[a]. Patricia Della Méa Plentz, Dr[a]
Coordenadora do Programa

Documento assinado digitalmente
Carlos Becker Westphall
Data: 23/08/2023 10:01:22-0300
CPF: ***.747.990-**
Verifique as assinaturas em https://v.ufsc.br

_____

Prof. Carlos Becker Westphall, Dr.
Orientador

Florianópolis, 2023.

Pedro Alexandre Barradas da Côrte

Análise do consumo de energia para dispositivo IoT no monitoramento dos recursos hídricos de larga escala

Dissertação submetida ao Programa de Pós-Graduação em Ciência da Computação para a obtenção do título de Mestre em Ciência da Computação.
Orientador: Prof. Carlos Becker Westphall, Dr.
Coorientador: Hugo Vaz Sampaio, Dr.

Florianópolis
2023

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

RESUMO

As redes de Internet das Coisas (IoT) são utilizadas para desenvolver soluções de automação em diversas áreas, como o monitoramento dos recursos hídricos de uma residência. Essas soluções têm um custo associado, e precisamos estar atentos às necessidades energéticas que elas apresentam. Propomos um estudo do consumo de energia de um sistema de gestão de água que é escalável, desde pequena até larga escala. Um dispositivo IoT de monitoramento do consumo de recursos hídricos composto por um Arduíno foi desenvolvido para servir de base para a análise inicial do consumo de energia e base para a rede de larga escala. Várias melhorias foram estudadas para reduzir o consumo de energia do dispositivo, e nossos resultados mostram até 69% de economia de energia. O consumo de energia da rede de larga escala também foi analisado em um contexto teórico, e melhorias também foram propostas. Possíveis estudos futuros sobre a densidade de dispositivos IoT na rede e como isso pode afetar o desempenho do sistema de gestão de água também foram apresentados.

Palavras-chave: internet das coisas. computação *fog*. casa inteligente. análise de energia. simulação de redes.

ABSTRACT

Internet of Things (IoT) networks are used to develop automation solutions in various areas, such as monitoring water resources in a home. These solutions have an associated cost, and we need to be aware of the energy needs they present. We propose a study of the energy consumption of a water management system that is scalable, from small to large scale. An IoT device for monitoring the consumption of water resources composed of an Arduino was developed to serve as a basis for the initial analysis of energy consumption and as a basis for the large-scale network. Several improvements have been studied to reduce device power consumption, and our results show up to 69% power savings. Large-scale grid energy consumption was also analyzed in a theoretical context, and improvements were also proposed. Possible future studies on the density of IoT devices in the network and how this can affect the performance of the water management system were also presented.

Keywords**:** *internet of things. fog computing. smart home. energy management. network simulation.*

DECLARAÇÃO DE BOLSA CAPES

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

1 INTRODUÇÃO

A evolução das tecnologias de comunicação e dos dispositivos inteligentes trouxe consigo a possibilidade de automatizar diferentes tarefas em diferentes áreas de trabalho ou pesquisa. O advento dos sistemas *Fog* permitiu que tais tarefas não se limitassem ao mesmo espaço onde ocorre a computação, trazendo soluções inovadoras para locais sem acesso a computadores. Alrawais et al. (2017) explicam que a *Fog Computing* possibilita uma nova geração de serviços na borda da rede. A única coisa necessária é uma conexão.

Diversas soluções estão focadas na automação de ambientes residenciais, com o objetivo de facilitar a vida das pessoas (BHILARE; MALI, 2016). De luzes que acendem com um comando verbal, a sistemas que controlam o uso de eletricidade de eletrodomésticos para reduzir custos, muitas ideias podem ser viáveis em casas inteligentes.

Um dos problemas de instalar essas diversas soluções é o rápido crescimento da quantidade de dispositivos que precisam se conectar à rede local utilizando o paradigma de Internet das Coisas (IoT), na maioria das vezes utilizando redes sem fio. Quando uma solução que adiciona uns 10 dispositivos sem fio em um apartamento, por exemplo, é instalada no prédio todo, a quantidade de dispositivos pode rapidamente ultrapassar 200 unidades em um único prédio.

Esta dissertação propõe o dispositivo IoT e busca analisar o consumo de energia de um sistema de gerenciamento de água de larga escala que utiliza o dispositivo em sua solução. O dispositivo precisa ser programado corretamente para minimizar o impacto energético que ele é capaz de causar quando utilizado em grande número.

O aumento da quantidade de dispositivos resultará em uma rede de larga escala que pode vir a sofrer problemas de performance devido à densidade de dispositivos IoT na mesma. Sendo assim, uma proposta de estudo do impacto da densidade em um sistema de água de larga escala também será oferecida.

1.1 MOTIVAÇÃO E JUSTIFICATIVA

Este estudo foi desenvolvido para buscar aprimoramentos para o dispositivo IoT desenvolvido e sua futura implementação em redes de larga escala. A análise de energia também ajudará a aprimorar outros projetos do Laboratório de Redes e Gerência da UFSC, o LRG.

1.2 OBJETIVOS GERAIS

O objetivo principal deste trabalho é desenvolver um dispositivo IoT e analisar, de maneira teórica, uma rede de larga escala que interconecta centenas de cópias desse dispositivo IoT para gerir o consumo de água em diversas residências que fazem parte um mesmo ambiente (geralmente, um condomínio). Para esse fim, este estudo precisa de:

1.2.1 Objetivos específicos

- desenvolver um dispositivo IoT para servir de base para a rede de larga escala;
- analisar o consumo de energia do dispositivo para servir de base para o consumo da rede de larga escala;
- aumentar a quantidade de dispositivos IoT para expandir a análise de consumo para uma análise teórica de larga escala.

1.3 ESTADO DA ARTE

Dois conceitos importantes para este trabalho são *Fog Computing* e IoT. O *Fog Computing* permite descentralizar o gerenciamento e processamento de dados, realizando tais tarefas na rede local de sensores e atuadores implantados (IORGA et al., 2018). Ele pode ser definido como um modelo separado de computação em camadas, onde os recursos computacionais são escaláveis, ou seja, podem ser alterados para fornecer mais (ou menos) poder computacional conforme a necessidade da rede. A IoT, por outro lado, define todos os dispositivos e sensores que povoam sua rede, e sua difícil implementação advém da complexidade no gerenciamento dos dados que são gerados pelos sensores e atuadores presentes na rede. Os sistemas tradicionais de IoT, que em sua maioria utilizam processamento de dados na *Cloud*, sofrem de alta latência e heterogeneidade, e os dispositivos presentes em uma IoT possuem características inerentemente desafiadoras (como baixo poder computacional, baixas taxas de transferência e armazenamento de dados limitado), que afetam a experiência do usuário final e também a Qualidade de Serviço (QoS) (ALRAWAIS et al., 2017).

O dispositivo IoT desenvolvido neste trabalho é um medidor inteligente. Esses medidores funcionam da mesma forma que os medidores analógicos, com a principal diferença de que os inteligentes convertem as medições para dados digitais, que podem ser utilizados por computadores ou servidores. Esses medidores também possuem conectividade com redes locais, permitindo a comunicação com os dispositivos que farão uso dos dados, classificando-os como dispositivos IoT.

Muita pesquisa tem sido feita sobre esses medidores inteligentes, especialmente na área de controle do consumo de energia elétrica. Patel, Mody e Goyal (2019) propõem um sistema de medição de consumo de energia elétrica para residências na Índia, que é controlado por um Arduino e usa um *"GSM shield module"* para comunicação sem fio. Sampaio et al. (2021) explora a gestão autonômica de energia com computação em névoa, e apresenta como utilizar o controle do ciclo de trabalho dos dispositivos IoT para economizar energia, além de utilizar diversas fórmulas para uma análise teórica.

1.4 ORGANIZAÇÃO DA DISSERTAÇÃO

Foi realizada uma pesquisa e avaliação de trabalhos publicados, com o objetivo de alcançar um melhor entendimento dos conceitos envolvidos, desde as funcionalidades e estruturas dos sistemas IoT até o uso de simuladores de redes. Esta dissertação está organizada em 7 Capítulos principais, sendo esta introdução o Capítulo 1. O Capítulo 2 apresenta os conceitos fundamentais para o desenvolvimento desta dissertação, que são seguidos por uma revisão bibliográfica no Capítulo 3, que também acompanha alguns dos artigos referenciados que foram analisados com mais detalhes. O Capítulo 4 trata do desenvolvimento do dispositivo IoT e a análise de energia do mesmo, e o Capítulo 5 a seguir apresenta os resultados dos testes realizados no dispositivo IoT e os cálculos de energia resultantes da análise. Para expandir os testes, o Capítulo 6 apresenta a análise de energia de um sistema de água IoT de larga escala. Já a Seção 6.3 apresenta a continuação deste trabalho, o estudo do efeito da densidade em redes de larga escala, que será feito de maneira teórica e simulada. Por fim, o Capítulo 7 conclui esta dissertação, descrevendo também outros possíveis trabalhos futuros.

2 CONCEITOS FUNDAMENTAIS

Este Capítulo aborda os principais conceitos fundamentais para o desenvolvimento desta dissertação, incluindo a Internet das Coisas, ou IoT, a computação *Fog*, ou computação na névoa, e simuladores de redes. O primeiro tópico foi utilizados para o desenvolvimento de um dispositivo IoT que utiliza a *Fog*, enquanto o segundo tópico foi utilizado para realizar os estudos e análises de redes de larga escala. O terceiro tópico servirá especificamente para a análise da densidade de redes sem fio de larga escala.

2.1 INTERNET DAS COISAS (IOT) E COMPUTAÇÃO *FOG*

Um dos maiores desafios encontrados ao tentar implantar um sistema de Internet das Coisas (*Internet of Things*, IoT) é a complexidade no gerenciamento de dados que são gerados por sensores e atuadores presentes na rede. Os sistemas IoT tradicionais, que utilizam majoritariamente processamento de dados na nuvem, sofrem principalmente com alta latência e heterogeneidade (de dispositivos e tipos de dados) (ALRAWAIS et al., 2017).

Segundo os autores Alrawais et al. (2017), os aparelhos presentes em uma IoT possuem características inerentemente desafiadoras (como baixo poder computacional, baixas taxas de transferência e armazenamento de dados limitado), as quais afetam a experiência do usuário final e também a qualidade do serviço (*Quality of Service*, QoS).

Um dos usos mais discutidos para um sistema IoT, com uso de computação na névoa, é a automatização de ambientes de moradia, em especial casas. O sistema proposto por Bhilare e Mali (2016) utiliza uma aplicação *web* (funcionando na nuvem) em conjunto com o sistema IoT para controlar aparelhos inteligentes dentro de uma casa, além de utilizar tal aplicação para monitorar os gastos com eletricidade. O objetivo é permitir que o usuário faça *login* no sistema para checar o consumo elétrico em tempo real e manualmente controlar seus aparelhos locais.

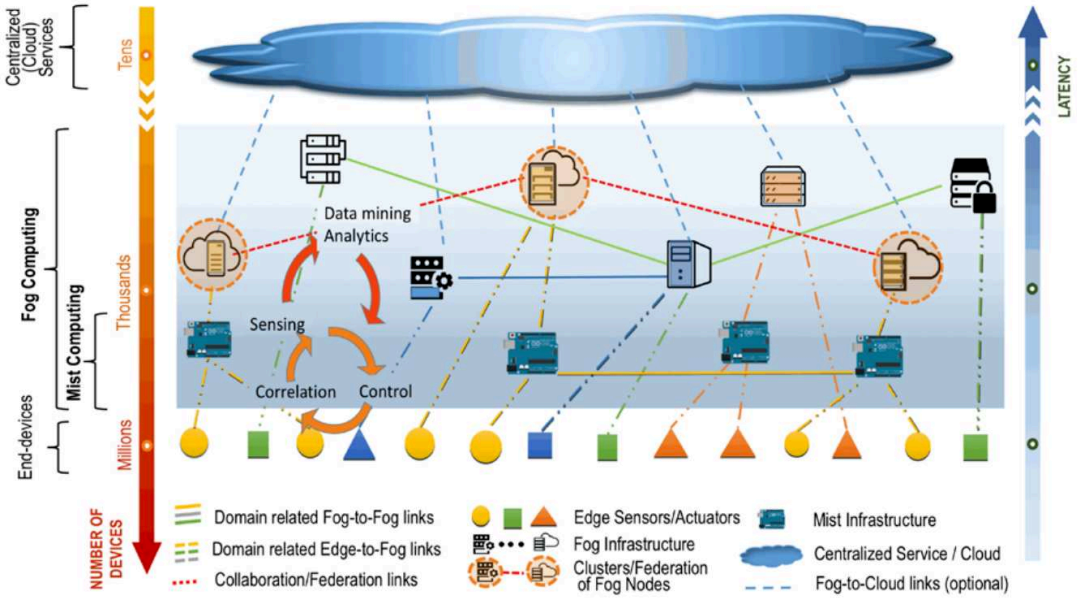
Um aspecto importante a ser considerado ao implantar um sistema IoT é o consumo de energia (corrente ou bateria) dos dispositivos presentes na rede (tanto os *smart devices* quanto os nós da névoa). Para a maioria dos sistemas IoT, é fundamental que os dispositivos implantados possam continuar ligados pela maior quantidade de tempo possível, o que pode se tornar difícil caso o consumo das baterias destes não sejam gerenciadas de forma efetiva (SAMPAIO et al., 2019).

A computação em névoa, ou *Fog Computing*, permite descentralizar o gerenciamento e o processamento de dados, executando tais tarefas na própria rede local dos sensores e atuadores implantados (IORGA et al., 2018). Este modelo ajuda a liberar recursos da nuvem, reduzindo a dependência de sistemas para todas as operações, além de resultar em um impacto positivo à qualidade de serviço e reduzir a latência da solução provida (ALRAWAIS et al., 2017).

O conceito *Fog Computing* pode ser definido como um modelo de computação separado em camadas, onde os recursos computacionais são escaláveis, ou seja, podem ser alterados para fornecer mais (ou menos) poder computacional, conforme as necessidades da rede. Na Fi-

gura 1, um modelo de *Fog Computing* é definido por Iorga et al. (2018), indicando as separações das camadas. Na camada superior situam-se os grandes centros de processamento, que possuem alto poder computacional e são responsáveis pela computação na nuvem (*Cloud Computing*). A camada intermediária, denominada *Fog*, encontra-se entre a *Cloud* e os dispositivos IoT.

Figura 1 – Descrição de camadas de um sistema *Fog*



Fonte: Iorga et al. (2018)

Na camada inferior, estão presentes os nós, que são, essencialmente, computadores que recebem e processam os dados dos sensores e atuadores da rede. Os nós *Fog* (tanto físicos quanto virtuais) são sensíveis ao contexto e utilizam todos um sistema comum para gerenciar e comunicar os dados. Eles podem ser organizados em *clusters*, com o objetivo de dar suporte ao isolamento (organizados de forma vertical), de dar suporte à federação (organizados de forma horizontal), ou até mesmo para reduzir a latência (organizados de forma relativa à latência entre os nós e os dispositivos da rede). Este modelo também pode fornecer, para os sensores, conectividade a serviços centralizados (IORGA et al., 2018).

2.2 TEORIA DE REDES DE FILAS

A teoria das filas é o estudo matemático das filas ou “filas de espera”, e é usada em vários campos, incluindo telecomunicações, ciências da computação, manufatura e saúde, para otimizar o design de sistemas e processos e melhorar sua eficiência e desempenho. Ela se preocupa em analisar e modelar vários tipos de filas e prever suas métricas de desempenho, como tempo médio de espera, número médio de “clientes” (pacotes) na fila e utilização do sistema. Toda esta seção foi escrita de acordo com Zukerman (2022).

Os tipos de fila são geralmente abreviados usando a notação de Kendall, que é mais utilizado em modelos de fila única seguindo o esquema $P/D/N/T-F$:

- P - Processo de chegada
- D - Distribuição do serviço
- N - Número de servidores
- T - Tamanho do *buffer*
- F - disciplina da Fila

As duas primeiras letras representam as distribuições do intervalo de tempo do processo de chegada e do tempo de serviço, e os valores mais comuns incluem *D* (Distribuição Determinística), *M* (Modelo Markoviano), *G* (Distribuição Geral), *GI* (Distribuição Geral e Independente) e *Geom* (Distribuição Geométrica). O modelo Markoviano usa Poisson para o processo de chegada e Exponencial para a distribuição do tempo de atendimento requerido por cada pacote, e estes dois valores são independentes entre si. As filas Markovianas também são *memoryless*, ou seja, a probabilidade de um pacote chegar ou sair da fila em qualquer período de tempo não será influenciada pelos eventos anteriores.

Filas do tipo $M/M/c/\infty$ serão mais utilizadas. Comparando essa notação com o esquema descrito acima, o primeiro *M* refere-se à distribuição dos tempos de chegada entre pacotes, que se assume exponencial, e o segundo *M* refere-se à distribuição dos tempos de atendimento, que também se assume exponencial. O número c ($c = 1, 2$, etc...) representa o número de servidores no sistema, e o símbolo ∞ representa o tamanho dos *buffers* de todos os servidores, que serão infinitos.

$$E\{T\} = \frac{1}{\mu - \lambda} \tag{1}$$

A Equação 1 será a mais relevante para calcular o tempo de espera e o processamento de um pacote, pois ela resulta no tempo de espera médio (T_E) de filas do tipo $M/M/1/\infty$, ou seja, aquelas que contém apenas um servidor. As variáveis μ e λ representam o tempo de saída e a taxa de chegada dos pacotes na fila, respectivamente. A razão entre essas duas variáveis é chamada de "rô", e pode ser observada na Equação 2.

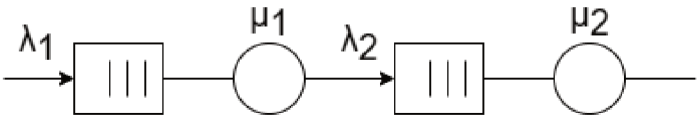
$$\rho = \frac{\lambda}{\mu} \tag{2}$$

Podemos considerar dois tipos de redes de filas, as Redes Fechadas e as Redes Abertas. Nas redes fechadas, os pacotes enviados de fora da rede não chegam nela, e os enviados dentro da rede não saem dela. Já as redes abertas possuem filas que atendem pacotes vindos de fora da rede, e os mesmos são enviados para fora da rede depois de serem atendidos. Essas redes abertas também podem ser separadas em dois tipos, aquelas Sem Realimentação e aquelas Com Realimentação.

As redes abertas sem realimentação utilizam o teorema de Burke, o qual estabelece que a saída de cada fila em série, cadeia ou cascata (como na Figura 2) com entrada Poisson e

serviço exponencial (ou seja, filas $M/M/...$) também possuem saídas Poisson, podendo assim ser considerado que as filas são independentes entre si. Devido a essa independência, em uma fila em série com uma taxa de chegada λ e tempo de serviço exponencial $\frac{1}{\mu}$, é possível calcular o tempo de atraso de cada nó individualmente, e a soma de todos os atrasos desses nós será o tempo total de atraso das filas.

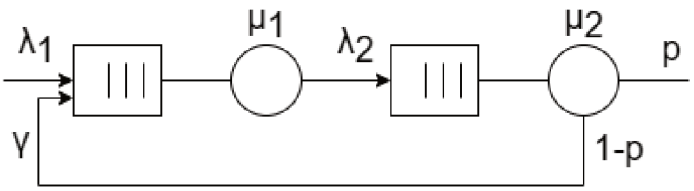
Figura 2 – Exemplo simples de filas em série



Fonte: Adaptado de Sampaio e Motoyama (2017)

Já as redes abertas com realimentação utilizam o teorema de Jackson, o qual estabelece que uma rede de filas com *feedback* (a realimentação em si) com entrada Poisson e serviço exponencial (ou seja, filas $M/M/...$) terá a sua taxa de *feedback* representada por γ. Um exemplo pode ser observado na Figura 3.

Figura 3 – Exemplo simples de filas com realimentação



Fonte: Adaptado de Sampaio e Motoyama (2017)

2.3 TEOREMA DE SHANNON

O teorema de Shannon foi publicado em Shannon (1948) e apresenta a relação observada entre a taxa máxima de transmissão de uma rede e a largura de banda da mesma (tal taxa de transmissão é considerada sem erros). O teorema utiliza a Equação 3 para afirmar que a capacidade do canal, ou seja, a quantidade máxima de dados que pode ser transmitida por segundo, é afetada diretamente pela largura de banda e pelo nível de ruído.

$$C = B \times \log_2(1 + \frac{P}{N}) \tag{3}$$

De acordo com Ma (2021), as variáveis da Equação 3 são: C, a capacidade do canal em bits por segundo (*bps*); B, a largura de banda em *hertz*; P, a potência do sinal da rede em *watts*; e N, a potência do ruído presente no canal, também em *watts*.

$$Signal\ to\ Noise\ ratio = 10 \times log_{10}(\frac{P}{N}) \tag{4}$$

O valor de $\frac{P}{N}$ é chamado de *SNR* (*Signal-to-noise ratio*, ou Relação Sinal-Ruído). De acordo com Peterson e Davie (1996), esse valor pode ser medido em decibéis (dB) utilizando a Equação 4. Por exemplo, para $\frac{P}{N} = 100$, temos um valor $SNR = 20\ dB$, como podemos observar na Equação 5. Da mesma forma, com um valor $SNR = 25\ dB$, temos $\frac{P}{N} = 316,228$, como podemos observar na Equação 6.

$$10 \times log_{10}(100) = 10 \times 2 = 20\ dB \tag{5}$$

$$25\ dB = 10 \times 2,5 = 10 \times log_{10}(10^{2,5}) = 10 \times log_{10}(316,228) \tag{6}$$

O teorema de Shannon implica no fato que o aumento da densidade de antenas em um sistema *wireless* pode causar atrasos no envio e a necessidade de retransmissões de pacotes. Isso porque esse aumento da densidade aumenta a interferência entre as mesmas, reduzindo a relação sinal/ruído ($\frac{P}{N}$) e, consequentemente, a capacidade (C) do canal.

3 REVISÃO BIBLIOGRÁFICA

Com o objetivo de salientar a necessidade e importância da pesquisa deste trabalho, uma revisão sistemática foi desenvolvida, utilizando três ferramentas de pesquisa acadêmica, sendo estas o Google Scholar, o IEEE Xplore™ e o ScienceDirect. As palavras-chave (em Inglês) foram pesquisadas nestas ferramentas tanto separadas quanto agrupadas, conforme mostrado na Tabela 1, que apresenta o número de ocorrências destas em cada ferramenta. Os artigos pesquisados foram limitados para apenas aqueles publicados após o ano 2018.

Tabela 1 – Resultados da revisão sistemática bibliográfica (atualizados em 9 de Maio de 2023)

Palavras-chave	Google Scholar	IEEE Xplore™	ScienceDirect
IoT	371.000	56.090	37.209
Fog	37.500	3.587	3.482
SH	62.700	3.378	7.485
EM	102.000	11.825	23.405
NS	19.600	1.003	3.003
IoT + *Fog*	28.300	1.795	2.919
IoT + SH	52.200	1.530	4.349
IoT + EM	24.900	714	3.040
IoT + NS	7.340	69	530
Fog + SH	10.200	40	872
Fog + EM	6.210	37	418
Fog + NS	1.240	5	100
SH + EM	16.900	342	1.950
SH + NS	1.080	3	121
EM + NS	1.710	7	182
IoT + *Fog* + SH	9.960	32	842
IoT + *Fog* + EM	5.680	20	388
IoT + *Fog* + NS	1.020	3	82
IoT + SH + EM	12.400	75	978
IoT + SH + NS	886	0	99
IoT + EM + NS	770	0	75
Fog + SH + EM	2.370	1	215
Fog + SH + NS	215	0	23
Fog + EM + NS	139	0	15
SH + EM + NS	267	0	37
IoT + *Fog* + SH + EM	2.320	1	207
IoT + *Fog* + SH + NS	211	0	23
IoT + *Fog* + EM + NS	143	0	14
IoT + SH + EM + NS	204	0	25
Fog + SH + EM + NS	57	0	9
IoT + *Fog* + SH + EM + NS	57	0	9

As palavras foram identificadas pelas seguintes siglas:

1. IoT - *Internet of Things* (pesquisada como “iot”)
2. *Fog* - *Fog computing*
3. SH - *Smart Home*
4. EM - *Energy Management*
5. NS - *Network Simulation*

Os dados da Tabela 1 ressaltam uma clara falta de artigos nos últimos cinco anos que tratem de *Fog Computing*, em especial quando combinada com as outras palavras-chave, sendo a combinação de *Fog Computing* com *Smart Home* o menor resultado do nível dois (o nível com duas palavras-chave). Nos resultados da ferramenta IEEE Xplore™, a principal fonte dos trabalhos referenciados nesta dissertação, a quantidade de artigos nos níveis três ou maior são extremamente limitadas, reforçando a necessidade de novas pesquisas envolvendo tais assuntos.

As palavras-chave *“Internet of Things”* e *“Fog computing”* representam a maior parte dos trabalhos que tratam do lado mais técnico da proposta, e sua utilização nos termos de pesquisa ajuda a filtrar quaisquer trabalhos sobre as outras palavras-chave que não utilizem tecnologia em suas soluções. *“Smart Home”* foi utilizada na revisão para limitar os tipos de soluções encontradas a ambientes menores, porém esta palavra-chave não é um fator extremamente necessário nos trabalhos relacionados. *“Energy Management”* é uma palavra crucial para a revisão deste trabalho, pois trata de um dos objetivos principais do mesmo. O controle de energia pode ser feito de diversas formas, incluindo aquelas que utilizam soluções desprovidas de IoT ou *Fog*, e é por isso que nenhum trabalho relacionado foi escolhido com base apenas nesta palavra-chave. Por fim, a palavra *“Network Simulation”* nos permitiu encontrar alguns trabalhos para fundamentar a próxima etapa desta dissertação, a qual dependerá de simuladores para que seja viável.

As combinações de palavras-chave mais úteis na revisão sistemática bibliográfica foram as de nível três, e a palavra *“Internet of Things”* foi a mais importante na pesquisa.

3.1 TRABALHOS RELACIONADOS

Serão agora apresentados os principais trabalhos relacionados considerando os tópicos mais importantes para esta dissertação, sendo estes a gerência de água, a gestão autonômica e o gerenciamento de energia. Também foram selecionados dois trabalhos que tratam de simulação de redes utilizando software. Todos os trabalhos escolhidos e apresentados utilizam IoT, fato que pode ser observado na Tabela 2 ao final deste Capítulo, onde os mesmos são comparados.

Os autores Aggarwal, Chauhan e Prakash (2019) propuseram um sistema inteligente de monitoramento de consumo de água em edifícios, abordando o desafio de altos custos de instalação de hidrômetros individuais. Eles desenvolveram um dispositivo econômico usando

a plataforma Arduino, equipado com sensores e módulos para coletar dados de consumo de água de diferentes apartamentos. O sistema automatizado, acessível e baseado em IoT, rastreia e monitora o consumo de água em tempo real, com sensores geograficamente distribuídos para capturar dados específicos de cada local do edifício. Esses dados são analisados por uma Rede Neural Artificial para gerar resultados precisos e imediatos. O sistema notifica os usuários sobre consumos excessivos através de um aplicativo Web e Android, permitindo que proprietários de edifícios gerem contas mais precisas e moradores identifiquem oportunidades para reduzir o consumo.

O dispositivo IoT utiliza o Arduino Mega para controlar diversos sensores simultaneamente, proporcionando uma alternativa mais acessível que outros microcontroladores disponíveis. Ele é alimentado por um painel solar e bateria, garantindo independência energética. A presença de sensores de fluxo, umidade, temperatura, relógio em tempo real e conectividade WiFi e Bluetooth otimizam a coleta e transmissão de dados, reduzindo custos de fiação. O modelo de Rede Neural Artificial é empregado para analisar padrões de consumo e gerar resultados precisos rapidamente. Um aplicativo intuitivo permite aos usuários autenticados visualizar os dados coletados, identificar padrões e agir para reduzir o consumo, garantindo a usabilidade e segurança do sistema.

Narayanan e Sankaranarayanan (2019) destacam a importância da integração da tecnologia de informação e comunicação para resolver desafios no sistema tradicional de distribuição de água. Eles propõem a criação de redes inteligentes de distribuição de água (*Smart Water Distribution Network*, ou SWDN) para cidades inteligentes, que incluem reservatórios, estações de bombeamento, hidrantes e linhas de serviço ao consumidor. Essas redes empregam tecnologias avançadas para detectar fluxo, pressão e consumo de água, visando uma distribuição eficiente e aumento da eficácia do sistema por meio do uso de *loops* e redundâncias inteligentes. O sistema integra a gestão da distribuição de água com a iniciativa de Cidades Inteligentes, preenchendo também a necessidade de operação de monitoramento e proteção da rede por meio do SCADA (*Supervisory Control and Data Acquisition*, ou Controle de Supervisão e Aquisição de Dados).

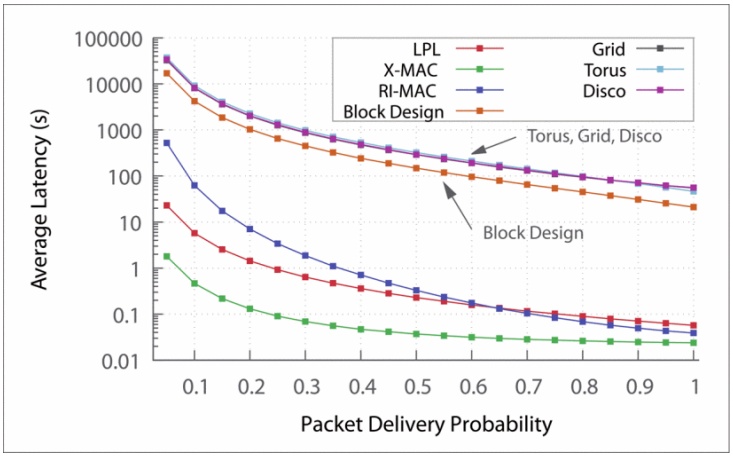
A proposta introduz uma SWDG (*Smart Water Distribution Grid*, ou Rede Inteligente de Distribuição de Água) para conectar os recursos hídricos e consumidores em cidades inteligentes. A responsabilidade de manutenção dos tubos é compartilhada entre o sistema supervisório SCADA e os proprietários, com base na localização dos tubos. O sistema proposto, baseado em IoT e monitoramento de integridade de dutos, é integrado com *Cloud* e *Fog*. A automação da distribuição de água oferece uma combinação de automação local e remota para controlar válvulas de fluxo e pressão, proporcionando eficiência, diagnóstico de falhas, regulação e controle da rede de distribuição. Esse sistema visa satisfazer as necessidades essenciais das cidades inteligentes, compreendendo a demanda de água, pressão, requisitos de fornecimento e detalhes da estrutura de distribuição.

Os autores Passos et al. (2019) exploram a questão do consumo energético de antenas em dispositivos IoT, propondo o controle do *Duty Cycle* (ciclo de trabalho) para otimizar o uso dessas antenas. Eles destacam desafios na coordenação de períodos ativos e inativos

das antenas, discutindo abordagens assíncronas para resolver esse problema. As abordagens *receiver-initiated* e *preamble sampling* são exploradas, juntamente com protocolos como RI-MAC (*Receiver-Initiated MAC*), NeWT (*Next Wake-up Time*), B-MAC, WiseMAC e RAW (*Random Asynchronous Wakeup Protocol*, ou protocolo de ativação assíncrona aleatória). Critérios como eficiência energética, latência, complexidade e custo são considerados nas avaliações, que incluem simulações para comparar o desempenho das abordagens.

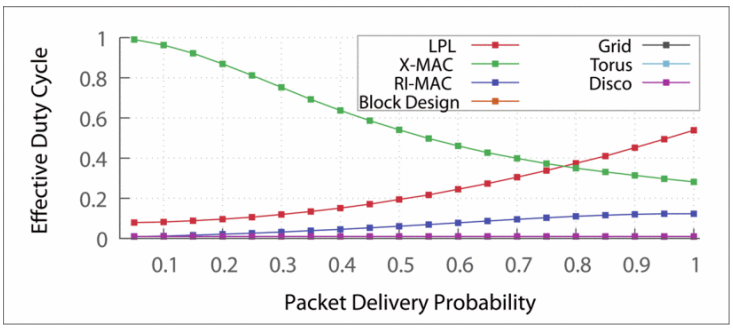
Apesar do progresso, os autores identificam desafios pendentes para a adoção ampla do ciclo de trabalho assíncrono, como melhorar a eficiência do rádio despertador, tratar a rejeição de ruído e interferência, ampliar o suporte a transmissão e *multicast*, e reavivar o desenvolvimento de abordagens aleatórias. As conclusões ressaltam a relevância contínua das abordagens assíncronas, apontando para futuras pesquisas. As Figuras 4, 5 e 6 fornecem *insights* sobre a latência, o ciclo de trabalho e o consumo de energia nas simulações.

Figura 4 – Latência *single-hop*



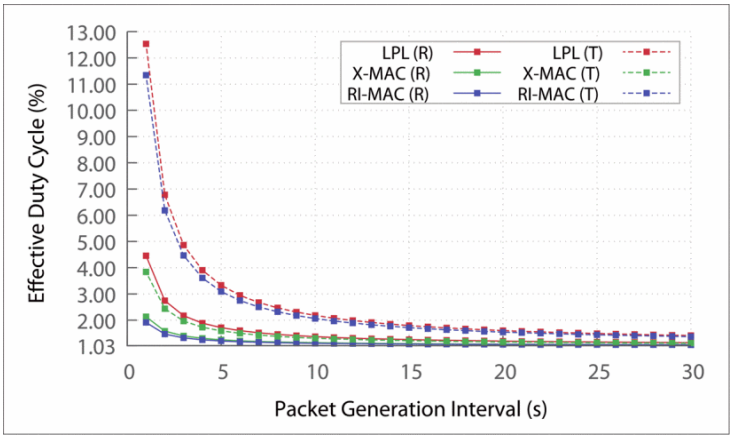
Fonte: (PASSOS et al., 2019)

Figura 5 – Ciclo de trabalho efetivo do receptor durante as simulações



Fonte: (PASSOS et al., 2019)

Figura 6 – Ciclo de trabalho efetivo para LPL, X-MAC e RI-MAC com intervalos de pacote diferentes entre receptor (R) e transmissor (T)



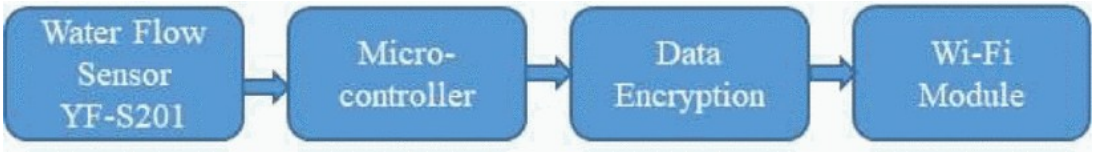
Fonte: (PASSOS et al., 2019)

No artigo dos autores Jeurkar et al. (2020), um sistema é descrito, no qual um medidor de vazão, sensor de temperatura e sensor de nível se conectam a um módulo WiFi através de um microcontrolador. Esses elementos permitem a medição precisa dos dados de vazão, temperatura e uso da água, bem como sua transmissão e exibição em smartphones através da plataforma IoT Thingsboard. Um alerta é ativado se a temperatura ultrapassar certo limite, enquanto o nível da água é mostrado graficamente. O sensor monitora constantemente o nível e dispara um alarme quando o tanque está cheio.

O medidor de vazão é instalado no tubo pelo qual a água flui em direção ao tanque, permitindo monitorar o fluxo para calcular o consumo total de água. Destaque é dado ao sensor de nível baseado em capacitância, onde um eletrodo no centro do tanque atua como uma das placas de um capacitor, e a água no tanque atua como a segunda placa. À medida que o nível de água muda, a capacitância do sensor é alterada, permitindo monitoramento sensível e contínuo.

O artigo Ray e Ray (2020) propõe um sistema seguro e inteligente de gestão de recursos hídricos com aplicação de *Machine Learning* na *Cloud*. Integrando nós *Fog*, o sistema pode ser implementado em larga escala. Para segurança, o sistema emprega o AES (*Advanced Encryption Scheme*, ou Esquema de Encriptação Avançado) para criptografar e descriptografar dados, enquanto o uso de nós *Fog* prioriza o envio de informações criptografadas para a *Cloud*.

Figura 7 – Esquema da unidade de aquisição de dados



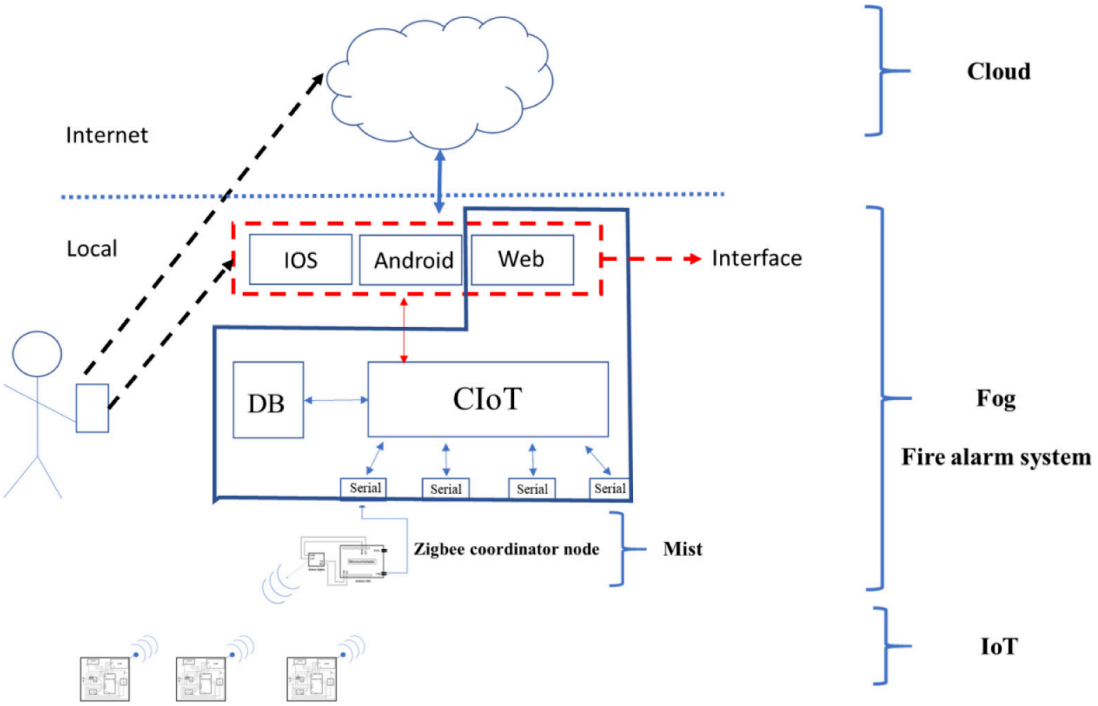
Fonte: (RAY; RAY, 2020)

O sistema de Ray e Ray (2020) inclui uma unidade de aquisição de dados com Arduino, sensor de fluxo de água e antena WiFi. Seu esquema de aquisição de dados foi apresentado na Figura 7. O nó *Fog*, que é um Raspberry Pi, coleta dados dos nós *Edge*, decide quais

dados enviar à *Cloud* e aplica análises de *Machine Learning*, incluindo regressão logística, para identificar padrões de uso de água. A plataforma *Cloud ThingSpeak* é usada para monitorar o consumo total de água e emitir alertas através do IFTTT WebHooks. O sistema tem potencial para implementar um modelo de receita de água, permitindo cobrança monetária automática com base no uso monitorado de cada domicílio.

O trabalho de Sampaio et al. (2021) apresenta a CIoT (*Central IoT*) como uma solução para otimizar o consumo de energia em dispositivos IoT, utilizando o *Fog Computing* para controlar ciclos de trabalho com base nas variações de ambiente e no comportamento dos residentes de um condomínio inteligente. A CIoT ajusta os ciclos *H* (Home) e *A* (Away) guiada por informações contextuais, como condições ambientais, comportamento do usuário e regulamentações energéticas, visando economia de energia. A implementação se concentra em um sistema de alarme de incêndio, atualizando seus sensores de acordo com o estado e a programação do usuário. A arquitetura proposta possui quatro camadas: *Cloud*, *Fog*, *Mist* e IoT, com o módulo central da CIoT gerenciando um aplicativo local e as comunicações com dispositivos IoT.

Figura 8 – Arquitetura da solução proposta para a CIoT



Fonte: (SAMPAIO et al., 2021)

Sampaio et al. (2021) conduziu experimentos para analisar a economia de energia, estimando o consumo de dispositivos em rede variando o tempo de *sleep* em ambientes domésticos e condomínios. Foram considerados 300 dispositivos de alarme de incêndio em um complexo de prédios, com ajuste nos ciclos *H* e *A*. A economia de energia alcançou 61,51% ao ajustar o ciclo *Long Sleep* para 4 segundos, destacando a oportunidade de economia ao equilibrar a qualidade de serviço e o tempo de *sleep*, com resultados não proporcionais ao aumento do ciclo

Long Sleep.

No trabalho de Kodali e Kirti (2020), a ferramenta NS-3 foi utilizada para simular uma rede IoT que foi criada especificamente para estudar simulações. O objetivo dos autores, além de criar e simular a rede IoT, é focar em estudar os resultados obtidos com a simulação e compreender como melhor utilizá-los. Tais resultados da ferramenta mostram a troca de pacotes de dados entre os nós, o tamanho dos pacotes, a hora em que um pacote é enviado ou recebido e os endereços IP e números de porta dos nós.

O projeto da rede IoT consiste em uma rede de sensores, um microcontrolador e armazenamento *Cloud*. Os autores afirmam que essa é uma arquitetura "bem comum" utilizada em aplicações como registro de temperatura, acúmulo de informações relacionadas à umidade, monitoramento de status de máquina ou dispositivo, dentre outras. As informações que precisam ser coletadas ou monitoradas são adquiridas de sensores cuja responsabilidade é sentir constantemente as condições do ambiente sob observação. Os dados obtidos dessa forma são enviados para armazenamento e observação para uma plataforma em *Cloud* com a ajuda de um microcontrolador habilitado para WiFi. Os transceptores de cada componente da rede baseada em IoT foram modelados usando NS-3.

O estudo de Oukessou, Baslam e Oukessou (2018) emprega a ferramenta NS-3 para comparar diretamente o *throughput* das redes IEEE 802.11ah (WiFi HaLow) e LoRaWAN, destacando as LPWANs (*Low Power Wide Area Networks*, ou Redes de Área Ampla de Baixa Potência) como fundamentais para a IoT devido à cobertura de longo alcance e uso de frequências abaixo de 1 GHz. A WiFi HaLow utiliza o mecanismo RAW para reduzir colisões em redes densas, enquanto a LoRaWAN emprega a modulação CSS (*Chirp Spread Spectrum*, ou Espectro de Propagação de Chilro[1]) e o SF (*Spreading Factor*, ou Fator de Propagação) para melhorar a robustez contra interferências e *multipath fading*.

Os resultados das simulações revelam que, para aplicações futuras que não necessitem de altas taxas de dados, a LoRaWAN é mais vantajosa. A taxa de congestionamento da WiFi HaLow atinge 56% com 750 nós sensores e uma perda de propagação de 50% a cerca de 1 quilômetro do ponto de acesso. Por outro lado, a LoRaWAN mostra perda de congestionamento abaixo de 37% para um *gateway* de acesso atendendo cerca de 8000 nós, com perda de propagação de 50% a aproximadamente 12,5 quilômetros. Dessa forma, Oukessou, Baslam e Oukessou (2018) conclui que a LoRaWAN é a melhor escolha para aplicações de IoT que requerem longo alcance com menor impacto de congestionamento na rede.

3.2 COMPARATIVO DOS TRABALHOS RELACIONADOS

Aqui apresentamos a Tabela 2, onde podemos ver os 8 trabalhos relacionados que foram apresentados. As características mais relevantes das soluções apresentadas por cada um dos trabalhos foram condensadas nas linhas da tabela, de forma a facilitar uma comparação

[1] Chilro: Voz aguda e gorjeada das aves; conjunto de pios.

dos mesmos com esta dissertação, que aparece na última coluna. A grande maioria dos trabalhos relacionados escolhidos utilizam IoT e sensores para monitoramento automático, porém a minoria deles leva problemas de redes, como a densidade, em consideração.

Tabela 2 – Comparativo dos trabalhos relacionados

Artigos		Aggarwal, Chauhan e Prakash (2019)	Narayanan e Sankaranarayanan (2019)	Passos et al. (2019)	Jeurkar et al. (2020)	Ray e Ray (2020)	Sampaio et al. (2021)	Kodali e Kirti (2020)	Oukessou, Baslam e Oukessou (2018)	Esta dissertação
Trabalha com:	IoT	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Fog	—	✓	—	—	—	✓	—	—	—
	Cloud	—	✓	—	—	—	—	—	—	—
	Sensores	✓	✓	—	✓	✓	✓	✓	—	✓
Possui monitoramento automático		✓	✓	—	✓	✓	✓	—	—	✓
Software	Código em Arduino	✓	✓	—	—	✓	—	—	—	✓
	Código em microcontrolador	—	—	—	✓	—	—	—	—	—
	Código em Raspberry Pi	—	—	—	—	✓	—	—	—	—
	"CIoT Framework"	—	—	—	—	—	✓	—	—	—
	Simulação em NS-3	—	—	—	—	—	—	✓	✓	✓
Permite controle manual por aplicativo		—	—	—	—	—	✓	—	—	—
Oferece solução para:	Casas inteligentes	✓	—	—	—	—	✓	—	—	✓
	Cidades inteligentes	—	✓	—	—	—	—	—	—	—
	IoT em geral	—	—	✓	—	—	—	✓	✓	—
	Agricultura em geral	—	—	—	✓	—	—	—	—	—
	Gerência de água em geral	—	—	—	—	✓	—	—	—	—
Medições	Volume de água	✓	✓	—	✓	✓	✓	—	—	✓
	Gastos de energia	—	—	—	—	—	✓	—	—	✓
	Temperatura e umidade	—	—	—	—	—	—	✓	—	—
	Throughput e perda de pacotes	—	—	—	—	—	—	—	✓	—
Considera problemas de redes		—	—	✓	—	✓	✓	✓	✓	✓

A primeira categoria analisada na Tabela 2 é com quais tecnologias o artigo trabalha, sendo estas a IoT, a *Fog*, a *Cloud* e os sensores inteligentes, que são dispositivos IoT específicos. Na sequência, analisamos quais trabalhos oferecem uma solução que monitora um sistema de maneira automática. A segunda categoria analisa quais *softwares* são oferecidos pela solução do artigo, sendo estes um código que é executado em Arduino, microcontrolador ou Raspberry Pi, uma framework CIoT (do trabalho de Sampaio et al. (2021)) ou um uso de simulações feitas

no NS-3. Na sequência, verificamos que apenas um artigo oferece controle ao usuário final por meio da solução de software utilizada. A terceira categoria analisa para que tipo de ambiente a solução oferecida será utilizada, sendo estes as casas e cidades inteligentes, aplicações de IoT de maneira geral, ou aplicações para agricultura ou gerenciamento de água em geral. A quarta e última categoria analisa que tipos de dados são medidos pela solução do artigo, sendo estes o volume de água, os gastos de energia, a temperatura e a umidade, e o *throughput* e perda de pacotes de rede. Na última linha analisamos quais artigos levam problemas de redes, como densidade e falhas de conexão, em consideração.

O trabalho de Aggarwal, Chauhan e Prakash (2019) apresenta bons resultados na gestão autônoma do consumo de água em apartamentos usando uma quantidade reduzida de medidores de água, enquanto o trabalho de Narayanan e Sankaranarayanan (2019) nos mostra uma estrutura de monitoramento complexa que ilustra até que ponto a ideia geral desta dissertação pode ser levada. Já o trabalho de Passos et al. (2019) explora o ciclo de trabalho, que é uma das ferramentas mais proeminentes para ajudar a reduzir o consumo de energia de futuras redes IoT. O trabalho de Jeurkar et al. (2020) apresenta um escopo semelhante ao dispositivo IoT desenvolvido para este estudo, e o trabalho de Ray e Ray (2020) essencialmente adiciona um aspecto de energia solar no mesmo, além de descrever o uso de serviços em nuvem para processamento de dados. O trabalho de Sampaio et al. (2021) explora um sistema IoT mais complexo, ao mesmo tempo em que fornece fórmulas úteis de consumo de energia. Por fim, os trabalhos de Kodali e Kirti (2020) e Oukessou, Baslam e Oukessou (2018) exploram a ferramenta NS-3 e simulam redes IoT utilizando-a, sendo que o primeiro descreve a utilização do simulador e como configurá-lo, enquanto o segundo utiliza-o para comparar duas redes *wireless* diferentes, testando o *throughput* e a perda de pacotes de cada uma.

Devido ao grande foco em consumo de energia nesta dissertação, o trabalho de Sampaio et al. (2021) ofereceu a base mais forte, pois tratou do consumo de energia de redes IoT utilizando os mesmos conceitos que Passos et al. (2019) apresentou. Esta dissertação evolui o foco de Aggarwal, Chauhan e Prakash (2019) e Jeurkar et al. (2020) em relação do sistema IoT, e aprimora os cálculos feitos em Sampaio et al. (2021). Os trabalhos de Kodali e Kirti (2020) e Oukessou, Baslam e Oukessou (2018) serviram de base para os trabalhos futuros que foram apresentados na Seção 6.3.

4 SISTEMA DE GERENCIAMENTO DE ÁGUA E DISPOSITIVO IOT

O conteúdo dos Capítulos 4, 5 e 6 foi submetido e aceito em Côrte et al. (2023).

Propomos um sistema que combina um sensor de vazão de água com um servidor, com o objetivo de monitorar e notificar automaticamente os usuários quando algo der errado, ou seja, os usuários devem ser notificados quando o consumo de água estiver próximo do limite mensal ou estiver muito alto quando em comparação com medições de dias (ou meses) anteriores, ou mesmo quando um vazamento pode estar ocorrendo. Para isso, o sensor de fluxo de água atualiza frequentemente suas leituras e as salva em um servidor local, que pode verificar os dados para alertar seus usuários sobre os cenários mencionados anteriormente.

O sistema inclui um sensor que pode ser acoplado a um duto de água na residência (de preferência na entrada), que pode ser visto na Figura 9. Este dispositivo faz medições de água várias vezes entre intervalos customizáveis e as envia para um servidor local por meio de uma rede local sem fio. A combinação do servidor e do dispositivo que faz essas medições junto com a rede, a qual permite a conexão dos dispositivos citados, resulta em uma rede IoT.

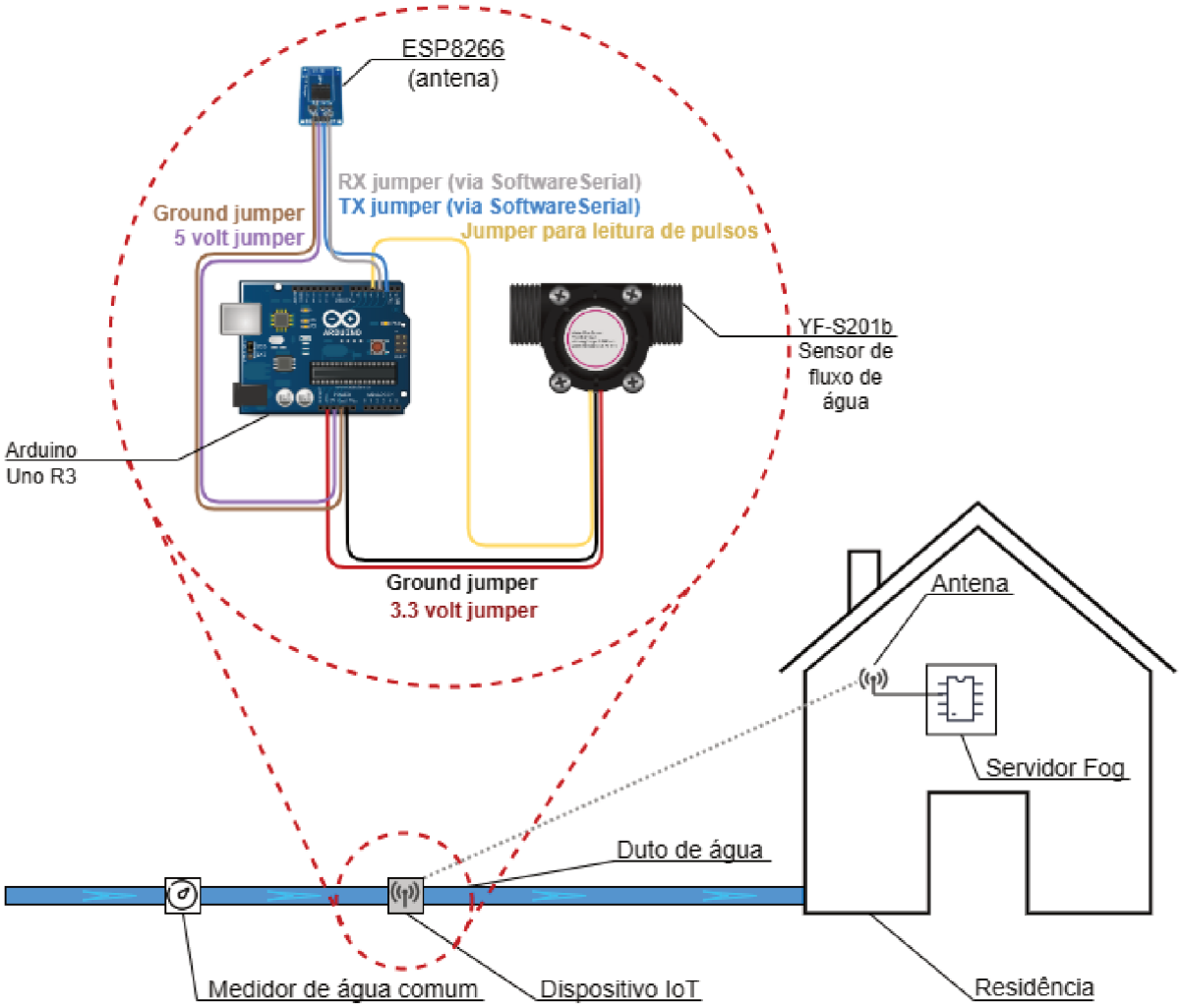
O servidor roda o sistema *Central Internet of Things* (ou CIoT), que foi desenvolvido por Sampaio et al. (2021) no Laboratório de Redes e Gestão (LRG) da UFSC, e esse sistema controla diversos subsistemas e dispositivos de uma casa inteligente, além de fornecer recursos para o sistema de monitoramento da água, como alertas de alto consumo (ou vazamento), disponibilidade de dados para análise do consumo médio, estimativa do custo da conta mensal de água de acordo com os padrões de consumo anteriores, "orçamento" para o uso da água com determinadas atividades, entre outros. Este servidor roda em um Raspberry Pi que contém um banco de dados onde as leituras do medidor são armazenadas, e a partir do qual é possível comparar novas leituras com leituras anteriores e manter um histórico das mesmas. Também é possível acessar um aplicativo por meio de um site hospedado no próprio Raspberry, por meio de qualquer computador ou smartphone conectado à internet.

O dispositivo IoT, exemplificado na Figura 9, é composto por um Arduino, um sensor de fluxo de água e uma antena. O Arduino lê as medições do sensor em cada intervalo de tempo configurado pelo usuário. Não é recomendado definir um intervalo de leitura muito curto, pois o aparelho se destina a ser instalado em locais que normalmente não possuem nenhum tipo de tomada elétrica próxima, o que significa que o sistema dependerá de bateria na maioria dos casos. A comunicação do Arduino com o servidor é feita através de uma conexão sem fio. Todos os dados obtidos pelo Arduino devem ser enviados apenas (em formato legível) ao servidor, e seu processamento lógico deve ser feito pelo servidor (conforme mencionado acima).

Com relação à energia da bateria mencionada, o sistema também deve procurar preservar o máximo de energia possível, minimizando o risco de erros de medição. Para isso, o Arduino deve manter o sensor de fluxo de água (que será o dispositivo com maior consumo de energia) desligado pelo maior tempo possível, cortando a energia de forma intermitente para reduzir o consumo geral. O período deve ser mínimo para evitar grandes erros de medição, mas não muito curto, caso contrário a economia não será suficiente para justificar este método.

Se for detectado fluxo de água, o sensor deve permanecer ligado durante esse período, o que significa que o sistema em geral durará mais em dutos de água raramente usados.

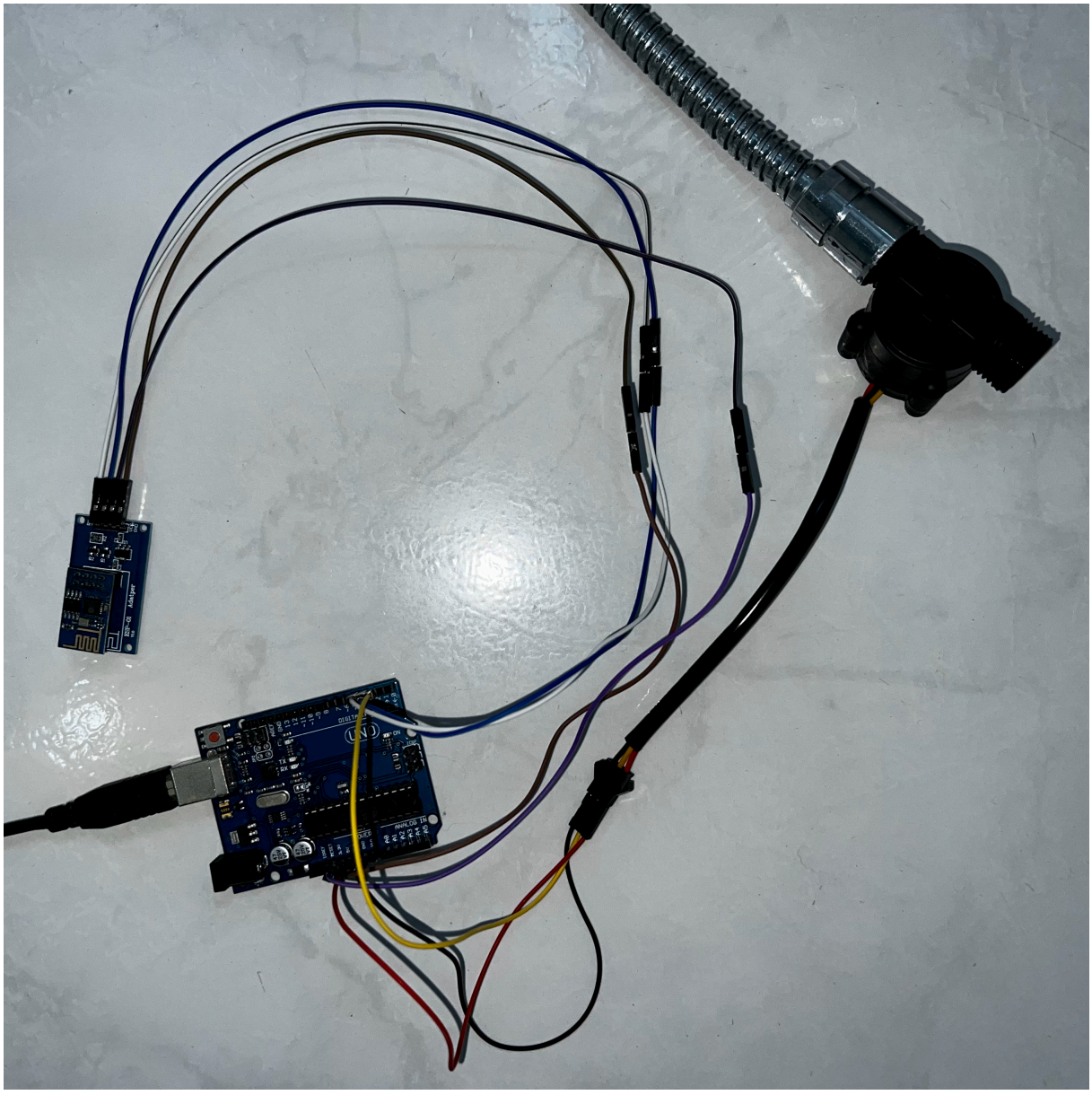
Figura 9 – Visão geral da proposta com fiação do protótipo



4.1 DESENVOLVIMENTO DE UM PROTÓTIPO

Para validar a proposta deste trabalho, foi desenvolvido um protótipo simples, composto por um Arduino, uma antena sem fio e um sensor de vazão de água. Todos esses componentes foram conectados através de cabos *jumper* (exemplificado na Figura 9 e executado na Figura 10) e, durante a simulação, o protótipo foi alimentado por um carregador de smartphone conectado à sua porta serial USB. O protótipo não leva em consideração as medidas de economia de energia. O sensor do protótipo foi conectado a uma mangueira de água para testar sua funcionalidade por meio de simulação de uso da água.

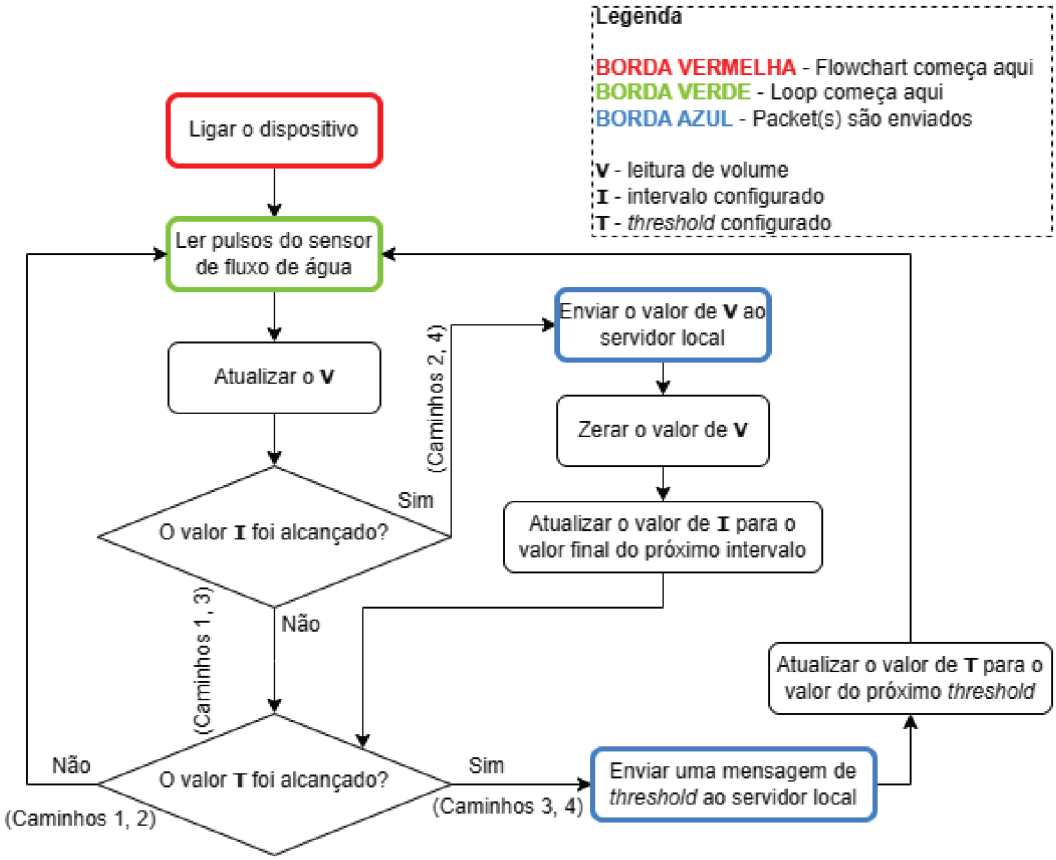
Figura 10 – Protótipo montado e conectado à mangueira



O sensor de fluxo de água YF-S201 foi utilizado. Este modelo limita-se a detectar de 1 a 30 litros de água por minuto e pode ser operado pelos pinos de 3,3 e 5 volts do Arduino. A antena usada no protótipo é a ESP8266, que utiliza conectividade WiFi de 2,4 GHz. Ele foi configurado (através do Arduino) para se conectar a uma rede sem fio local por meio de uma rede WiFi criada para o protótipo, cujos detalhes foram codificados no Arduino.

Existem dois valores que são customizáveis, sendo eles o intervalo de tempo de medição em que as leituras são salvas e um limite que produz um aviso sempre que seu valor é atingido, chamado de *threshold*. No entanto, esses valores, embora de fato "customizáveis", foram *"hardcoded"* no programa Arduino, tornando-os impossíveis de ajustar uma vez que o sistema comece a funcionar. Isso foi feito devido a limitações na comunicação do servidor para a antena do dispositivo IoT.

Figura 11 – O fluxograma (simplificado) do protótipo desenvolvido



O ciclo operacional do Arduíno e suas principais ações são ilustrados na Figura 11. Para permitir que as informações coletadas pelo Arduino sejam recebidas e armazenadas, um computador local foi configurado para executar um *script* Python que lê as informações transmitidas pela antena em uma porta de rede específica. As mensagens enviadas pela antena são codificadas em um formato específico. A antena envia duas mensagens em sequência, uma que informa o tamanho da próxima mensagem e outra que contém os dados relevantes. Isso é feito para maximizar o *buffer* da rede e evitar a perda de mensagens, o que pode quebrar o sistema.

Essas mensagens podem vir escritas como “m2345”, por exemplo. O primeiro carácter da mensagem que o Arduíno envia serve para identificar o tipo de mensagem sendo impressa, e é sempre uma letra (descritas mais abaixo). As mensagens também contém uma quantidade variada de números, às vezes com casas decimais separadas por um ponto, que podem representar medidas de volume ou tempo decorrido. As letras que definem os diferentes tipos de resposta são:

- Letra `s`: indica que os valores iniciais de intervalo de medição e aviso de *threshold* foram configurados, incluindo seus valores. O formato é `s1234s1234s`, onde as sequências `1234` representam os valores de intervalo de medição e aviso de *threshold*, respectivamente.
- Letra `m`: indica que uma nova leitura de volume está contida na mensagem. O formato é `m1234.56m`, onde os dígitos representam o volume em *ml*.

- Letra `t`: indica que um aviso de *threshold* foi lançado. O formato é `t01t02t03t`, onde os números entre cada `t` representam HORAS, MINUTOS e SEGUNDOS, respectivamente.

Tabela 3 – Tabela dos caminhos do fluxograma da Figura 11

Caminho	Probabilidade	Quando ocorre?	O que acontece?	Qual é o caminho no fluxograma?
1	Mais provável	Quando nem o intervalo nem o limite foram atingidos	O *loop* não faz nada	*No - No*
2	Segundo mais provável	Quando apenas o intervalo foi atingido	O volume do intervalo é enviado ao servidor	*Yes - No*
3	Terceiro mais provável	Quando apenas o limite foi atingido	O aviso de limite é enviado ao servidor	*No - Yes*
4	Menos provável	Quando tanto o intervalo quanto o limite foram atingidos no mesmo *loop*	Tanto o volume do intervalo quanto o aviso de limite são enviados ao servidor (nessa ordem)	*Yes - Yes*

4.2 ANÁLISE DE ENERGIA DO DISPOSITIVO IOT

É provável que haverá longos períodos em que este sistema não observará qualquer fluxo de água, o que significa que não reportará nenhum consumo. Como o sensor de fluxo de água YF-S201b não pode ser desligado por longos períodos sem incorrer na possibilidade de erros, pois não há como saber se há fluxo de água sem usar o próprio sensor, ele está constantemente consumindo eletricidade. Inspirada por Sampaio et al. (2021), a Tabela 4 foi criada para mostrar os valores de consumo de energia dos dispositivos usados no protótipo (tais valores foram obtidos de suas respectivas fichas de dados, YF-S201b (2021), ESP8266 (2021) e Arduino (2021), exceto onde outra fonte for indicada).

Tabela 4 – Consumo de energia do dispositivo

Dispositivo	Por hora		Por minuto		Voltagem
	mAh	Wh	mAh	Wh	
YF-S201b (constante)	15	0.075	0.25	0.00125	5V
ESP8266 (*Standby*)	0.9	0.003	0.015	0.00005	3.3V
ESP8266 (*Sleep*)	0.01	0.000033	0.00017	0.00000055	3.3V
ESP8266 (transmitindo)	Por volta de 173	Por volta de 0.571	Por volta de 2.883	Por volta de 0.0095	3.3V
Arduino (SAMPAIO et al., 2021)	Por volta de 0.3	Por volta de 0.0027	Por volta de 0.005	Por volta de 0.000045	9V

Agora estimamos o consumo de energia do sistema analisando quanta energia cada dispositivo consome, sendo o valor do sensor de fluxo de água por hora seu valor máximo e o valor de "transmissão" da antena sendo uma média dos quatro valores listados em sua ficha de dados (215, 197, 145 e 135, obtidos de (ESP8266, 2021)). Assumimos que a antena leva em média 10 milissegundos para enviar um pacote, deixando-a em modo de espera pelos 990 milissegundos restantes de cada segundo. Se utilizarmos a função *sleep* da antena, assumimos que ela teria que ficar em *standby* por pelo menos 5 *ms* antes e depois de enviar um pacote, deixando um total de 980 *ms* por minuto em modo *sleep*. Este comportamento da antena pode ser visto na Figura 12.

Para qualquer execução de n minutos (onde os dispositivos estão constantemente ligados), e considerando que a antena estará enviando os pacotes de dados em intervalos de 1 minuto (i), podemos desenvolver a Equação 7 onde:

- C_e é o consumo de energia do sensor de água por minuto;
- C_a é o consumo da antena em *standby* por minuto;
- C_b é o consumo de transmissão da antena por minuto;
- C_c é o consumo do Arduino por minuto; e
- C_{prot} é o consumo total do protótipo.

$$\sum_{i=1}^{n}(C_{e_i} + 5999\frac{C_{a_i}}{6000} + \frac{C_{b_i}}{6000} + C_{c_i}) = C_{prot} \tag{7}$$

Para contabilizar um minuto da antena considerando que ela não estará sempre no mesmo estado, foram somados 59990 milissegundos de C_a e 10 milissegundos de C_b, o que totaliza um minuto entre os dois valores da antena. Para reduzir o tamanho escrito das equações, os valores foram divididos por 10, o que converte-os para centissegundos, o equivalente de uma base de 10 milissegundos.

Agora, se o protótipo for programado para permitir que a antena entre no modo *sleep* (em vez de ficar em *standby*), poderíamos alterar a Equação 7 para obter a Equação 8 onde:

- C_s é o consumo da antena no modo *sleep* por minuto; e
- $C_{antenna}$ é o consumo total do protótipo teórico.

$$\sum_{i=1}^{n}(C_{e_i} + \frac{C_{a_i}}{6000} + \frac{C_{b_i}}{6000} + 5998\frac{C_{s_i}}{6000} + C_{c_i}) = C_{antenna} \tag{8}$$

Para contabilizar um minuto da antena considerando que ela não estará sempre no mesmo estado, foram somados 10 milissegundos de C_a, 10 milissegundos de C_b, e 59980 milissegundos de C_s, o que totaliza um minuto entre os três valores da antena.

Essas equações podem ser facilmente multiplicadas para estimar o consumo de qualquer quantidade de dispositivos IoT, o que permite estudar as possibilidades de expansão do sistema proposto. O valor de *n* minutos também pode ser aumentado continuamente para representar dias, meses e anos de consumo estimado.

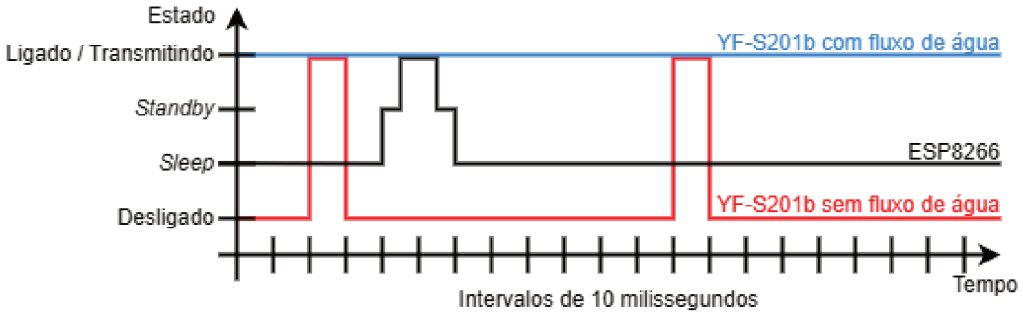
4.3 PROPOSTA DE CONTROLE DO DUTY CYCLE

Também podemos hipotetizar uma estratégia em que manteríamos o sensor de fluxo de água desligado a maior parte do tempo, ajudando a economizar mais energia em geral (SAMPAIO et al., 2021). De acordo com a folha de dados do sensor YF-S201b (2021), leva cerca de 0,04 microssegundos (μs) para o sensor enviar um sinal depois de ligado, o que significa que seu tempo de ativação é praticamente insignificante. Se o Arduino fosse conectado de uma maneira que permitisse cortar a energia (e restabelecê-la) ao sensor de fluxo de água sob comando, poderíamos desligá-lo continuamente por curtos períodos para economizar energia, permitindo que o sensor permanecesse totalmente ligado apenas quando um fluxo de água for detectado. Isso também significa que o sensor teria dois estados de funcionamento, *Active* (quando há fluxo de água) e *Idle* (quando não há fluxo de água).

O único problema com a abordagem proposta é que ela terá uma margem de erro, pois é possível que o fluxo de água inicie assim que o sensor de fluxo de água for desligado (ou a qualquer outro momento), o que significa que pode haver um período de fluxo de água que não é contabilizado tão longo quanto o sensor for mantido desligado. Como exemplo, e com base na ficha técnica do sensor de vazão de água YF-S201b (2021), se o sensor perder 50 *ms* de vazão de água, considerando que o volume máximo detectável é de 30 litros por minuto, haveria um erro de até 25 mililitros.

O consumo de energia pode ser calculado usando a Equação 8 para qualquer período de tempo em que haja fluxo de água, pois o sensor estará totalmente ativo durante esse tempo, que é o exato comportamento mostrado na equação (e assim, podemos dizer que $C_{antenna}$ é o consumo total do protótipo teórico quando há fluxo de água). No entanto, para qualquer período de tempo em que não haja fluxo de água, precisaremos de uma nova equação onde o consumo do sensor seja reduzido. A Equação 9 é apresentada, na qual a variável C_e é multiplicada por uma nova variável x. Essa nova variável é um número que varia de 0,01 a 1, e representa o tempo (em porcentagem) que o sensor fica acionado durante um minuto. Se observarmos a Figura 12, veremos que $x = 0,1$, pois o sensor de fluxo de água é ativado por 10ms a cada 100ms, ou seja, está ativo durante 10% de cada minuto. Por fim, C_{off} é o consumo total do protótipo teórico quando não há vazão de água.

Figura 12 – Ciclo de trabalho do protótipo para a proposta de energia



$$\sum_{i=1}^{n}(x \times C_{e_i} + \frac{C_{a_i}}{6000} + \frac{C_{b_i}}{6000} + 5998\frac{C_{s_i}}{6000} + C_{c_i}) = C_{off} \tag{9}$$

Agora podemos generalizar a maioria dos cenários combinando ambas as Equações 8 e 9 para obter a Equação 10. Esta nova equação possui algumas variáveis extras que contextualizam o período de n minutos que está sendo analisado, sendo elas:

- w, que é a quantidade de minutos em que o fluxo de água foi detectado, e substitui n na Equação 8; e
- z, que é a quantidade de minutos em que o fluxo de água não foi detectado, e substitui n na Equação 9.

É importante observar que, para qualquer execução de teste que dure n minutos, temos que $w + z = n$. Se necessário, as variáveis w e z podem ser qualquer número racional, o que permite que elas representem qualquer outra unidade de tempo menor, como milissegundos (embora ainda sejam medidas em minutos). Por fim, C_T é o consumo total do sistema desta proposta energética.

$$C_{antenna}^{n=w} + C_{off}^{n=z} = C_T \tag{10}$$

Vamos agora considerar um cenário onde o sensor de fluxo de água é mantido desligado por 90ms a cada 100ms, ativando-o para detectar o fluxo de água durante os 10ms restantes. Este comportamento é o mesmo discutido anteriormente (que pode ser visto na Figura 12), e também pode ser alterado remotamente se necessário. Significa efetivamente que, em qualquer período de tempo em que nenhum fluxo de água seja detectado, o sensor seria desligado por 90% do tempo, consumindo igualmente menos energia. Para nosso cenário, teríamos $x = 0,1$ (já que queremos que o sensor fique desligado por 90% do tempo), enquanto as outras duas variáveis dependem muito de um teste real para serem obtidas.

5 TESTES E RESULTADOS DO DISPOSITIVO IOT

Este Capítulo apresenta todos os testes feitos e resultados obtidos, tanto do protótipo desenvolvido na Seção 4.1, quanto das equações elaboradas nos Capítulos 4.2 e 4.3.

5.1 TESTES E RESULTADOS DO PROTÓTIPO

Com o código compilado e executando tanto no Arduino quanto no computador local, alguns testes foram realizados para validar o protótipo. Os testes principais foram feitos conectando o sensor de fluxo de água a uma mangueira de água em dois ambientes, um banheiro e um jardim. A torneira que controla o fluxo de água da mangueira que estava sendo usada foi aberta e fechada manualmente para simular um uso pseudo-aleatório e natural. Como precaução extra e para garantir a precisão, um copo medidor foi usado periodicamente durante os testes.

Os valores para o intervalo de tempo e aviso de limite foram mantidos em 1 minuto e 5 litros, respectivamente. Neste caso, a escolha de conectar o protótipo à mangueira do banheiro se deu por dois motivos, primeiro pela compatibilidade da mangueira com o diâmetro do sensor de vazão de água (ou seja, pela facilidade de conectá-los), e segundo porque foi destinado exclusivamente para fins de simulação.

O único objetivo dos testes é demonstrar que o protótipo funciona, e os resultados obtidos com ele não representam o uso real, embora permitam uma visão sobre o desempenho do protótipo. Para analisar os resultados, ao longo dos testes, a água que corria pelo sensor foi coletada em um pequeno copo medidor e o volume foi então comparado com as leituras do protótipo. Com isso, e analisando os resultados, foi possível concluir que o sensor de vazão de água utilizado mediu 25% a mais de volume do que realmente escoou, o que torna esse sensor inviável para aplicações de alta precisão, porém não inviabiliza a ideia do protótipo como um todo. Houve também um pequeno atraso de milissegundos entre as leituras que, embora não seja perceptível na forma como os dados são armazenados pelo *script* (já que este não mantém registros de data e hora), contribui para uma mudança geral no momento em que as leituras são feitas, o que pode ter um impacto negativo a longo prazo.

Com os dados produzidos pelo Arduino, o servidor mencionado na proposta pode utilizar os valores (e *timestamps*, caso sejam salvos no futuro) para habilitar todas as outras funções também mencionadas anteriormente. Esses dados permitem que o servidor gere continuamente um gráfico de consumo que pode permitir ao servidor buscar vazamentos identificando leituras irregulares (que, na perspectiva do servidor, seriam leituras que não seguem o padrão comum de uso de água da residência de acordo com o histórico de consumo) e também gerar alertas de limites mensais de consumo (simplesmente somando todas as leituras de um determinado mês). A variável de limite personalizável do Arduino permite que objetivos menores sejam configurados, ajudando a aprimorar os recursos do servidor.

5.2 ANÁLISE DE ENERGIA

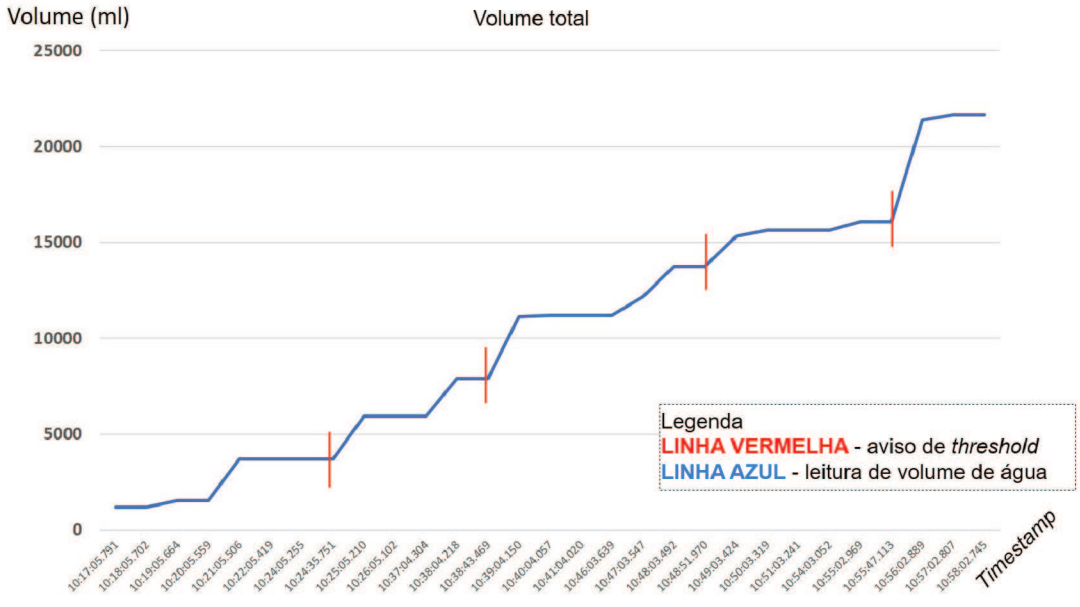
Faremos dois conjuntos de cálculos, um onde consideraremos um cenário onde baterias AA são usadas para alimentar o sistema e outro que analisa o consumo em *watts* por hora (inspirado por Sampaio et al. (2019) e Sampaio et al. (2021)). Esses dois conjuntos nos permitirão analisar o sistema independentemente de qual fonte de energia ele usaria na versão final. Para o primeiro, consideramos uma bateria AA com capacidade de 2000 miliamperes por hora (mAh) e tensão de 1,5 volts. Podemos usar seis pilhas AA em série para produzir o requisito de 9V do Arduino.

Na Equação 7, a antena ficará em *standby* por 59990 *ms* e transmitindo por 10 *ms* para cada minuto de execução. Estimamos que o sistema consumiria C_{prot} = 16,229 mAh, ou 0,08079 Wh. Este sistema pode funcionar por cerca de 123,239 horas com baterias.

Na Equação 8, a antena só ficará em *standby* nas janelas de 5 *ms* antes e depois de seu período de transmissão de 10 *ms*, deixando os 59980 *ms* restantes de cada minuto de execução para o estado de *sleep*, que consome bem menos energia. Estimamos que o sistema consumiria $C_{antenna}$ = 15,339 mAh, ou 0,07783 Wh. Este sistema pode funcionar por 130,385 horas com baterias.

Na Equação 9, para cada minuto de tempo de execução, a antena se comportará exatamente como na Equação 8; no entanto, o sensor de fluxo de água consumirá apenas uma porcentagem x de energia, de acordo com o tempo que estiver ligado a cada minuto. Se usarmos o valor discutido anteriormente, onde $x = 0,1$, em nosso cenário de baterias AA, o sistema consumiria C_{off} = 1,839 mAh (ou 0,01033 Wh), permitindo que funcione por impressionantes 1087,443 horas em baterias, o que só é possível porque o sistema não faria quase nada durante todo esse tempo, já que a Equação 9 representa o cenário "sem fluxo de água".

Figura 13 – Gráfico de dados gerado a partir dos resultados do teste



Para a Equação 10, tomaremos o cenário apresentado pela Figura 13, onde $w = 13$ e $z = 47$ (e assim temos um período de $n = 60$), pois podemos contar um total de 47 minutos em que não houve fluxo de água no gráfico. Estimamos que o sistema consumiria C_T = 4,764 mAh para esta hora de execução, ou 0,02495 Wh. Se tivéssemos 13 minutos consistentes de fluxo de água por hora, ou seja, $w = 13$ para cada $n = 60$, o sistema funcionaria por 419,8 horas com baterias.

Todos os resultados obtidos com as Equações 7, 8, 9 e 10 são apresentados na Tabela 5, que também apresenta o consumo do dispositivo em Wh, uma vez que essa métrica pode ser obtida usando as mesmas equações.

Tabela 5 – Os resultados para todas as equações

Equação	Variáveis	Consumo mAh	Duração da bateria de 2000 mAh (em horas)	Ganhos de longevidade	Consumo Wh	Economia de energia
7	$n = 60$	16,229	123,239	0%	0,08079	0%
8	$n = 60$	15,339	130,385	+5,8%	0,07783	+3,67%
9	$n = 60$ $x = 0,1$	1,839	1087,443	+782,39%	0,01033	+87,22%
10	$n = 60$ $x = 0,1$ $w = 13$ $z = 47$	4,764	419,800	+240,64%	0,02495	+69,11%

Tabela 6 – Os resultados estimados de Wh ao alterar x na Equação 10

x	Wh	$E\%$	$E_x(\%)$	$M_{spec}(ml)$	$M_{test}(ml)$	$M_w(ml)$
0,01	0,01967	75,66	74,73	29700	4950	386100
0,1	0,02495	69,11	67,94	27000	4500	351000
0,2	0,03083	61,84	60,39	24000	4000	312000
0,3	0,03670	54,57	52,84	21000	3500	273000
0,4	0,04258	47,30	45,29	18000	3000	234000
0,5	0,04845	40,03	37,74	15000	2500	195000
0,6	0,05433	32,76	30,19	12000	2000	156000
0,7	0,06020	25,49	22,65	9000	1500	117000
0,8	0,06608	18,21	15,10	6000	1000	78000
0,9	0,07195	10,94	7,55	3000	500	39000
1,0	0,07783	3,67	-	0	0	0

Podemos ver na Tabela 5 que todas as equações após a Equação 7 fornecem alguma economia de energia em comparação com ela. Como a Equação 8 só aplica o ciclo de trabalho da antena e sempre permite que o sensor de fluxo de água esteja ligado, ela oferece a menor economia. Por outro lado, a Equação 9 fornece uma tremenda economia de energia de 87%, mas isso se deve principalmente à situação de "sem fluxo de água" mencionada anteriormente. Finalmente, a Equação 10 mostra quase 70% de economia de energia quando comparada com a Equação 7. Se tomássemos a Equação 8 como nosso ponto de partida, uma vez que reflete

um cenário de "fluxo de água ininterrupto", as Equações 9 e 10 mostrariam uma economia de energia de 86,73% e 67,94%, respectivamente (esses dois valores não estão na Tabela 5).

A Tabela 6 explora o que acontece quando alteramos o valor de x na Equação 10 dentro de seu intervalo de 0,01 a 1, já que a porcentagem x de consumo de energia pode ter um impacto tanto na longevidade quanto na taxa de erro possível (e, posteriormente, na margem de erro). As últimas 6 colunas da Tabela 6 são representadas por variáveis, que significam o seguinte:

- Wh representa a quantidade de *watts* por hora que são consumidos em um minuto;
- $E\%$ representa a economia de energia comparado à Equação 7;
  - $E\% = |\frac{Wh \times 100}{0{,}08079} - 100|$
- $E_x\%$ representa a economia de energia extra quando comparado a não controlar o sensor de fluxo de água (ou seja, quando comparado à Equação 8);
  - $E_x = |\frac{Wh \times 100}{0{,}07783} - 100|$
- M_{spec} representa o erro máximo em *ml* por minuto com base na vazão máxima do sensor (30 litros por minuto ou 30000 *ml* por minuto, de acordo com YF-S201b (2021)), que pode ser alcançado se todo o fluxo de água ocorrer durante o tempo em que o sensor estiver desligado;
  - $M_{spec} = |30000 \times (x-1)|$
- M_{test} representa o erro máximo em *ml* por minuto com base na vazão máxima atingida durante o teste, que é de cerca de 5 litros por minuto ou 5000 *ml* por minuto;
  - $M_{test} = |5000 \times (x-1)|$
- M_w representa M_{spec} aplicado especificamente ao nosso cenário de teste, onde foram detectados 13 minutos de fluxo de água (o valor de w neste cenário).
  - $M_{wtest} = M_{test} \times w$

Para consistência, temos $w = 13$ e $z = 47$ para todos os cálculos da Tabela 6, o mesmo que a Equação 10 na Tabela 5. Podemos ver na Tabela 6 que alterar o valor de x afeta diretamente a economia de energia ($E\%$) e erros máximos (M_{spec}, M_{test} e M_w). Ao usar períodos de ativação mais curtos por minuto (valores menores de x), a economia de energia é maior, mas os erros máximos que podem ocorrer também são maiores, o que não é bom. Ao usar períodos mais longos (valores maiores de x), embora tenhamos erros máximos muito menores, também perdemos economia de energia. Portanto, não há valor correto para x, pois depende muito das necessidades do usuário final.

5.2.1 Analisando períodos maiores

Depois de usar a Equação 10 com os valores fixos para nosso cenário de teste, podemos expandir seu uso manipulando os valores de w e z para estimar a economia de energia de qualquer cenário. Apresentamos a Tabela 7, onde testamos vários valores para n, x, w e z para criar cenários representando dias, meses e anos. Todos os cálculos nesta tabela derivam especificamente da Equação 10, e as variáveis são as mesmas da Tabela 6.

Tabela 7 – Os valores estimados de Wh ao alterar todas as variáveis da Equação 10

n	x	w	z	Wh	$E\%$	$E_x(\%)$	$M_{spec}(ml)$	$M_{test}(ml)$	$M_w(ml)$
60	0,1	13	47	0,02495	69,11	67,94	27000	4500	351000
60	0,8	13	47	0,06608	18,21	15,10	6000	1000	78000
1440	0,1	312	1128	0,59888	69,11	67,94	27000	4500	8424000
1440	0,8	312	1128	1,58588	18,21	15,10	6000	1000	1872000
10080	0,1	2184	7896	4,19219	69,11	67,94	27000	4500	58968000
10080	0,8	2184	7896	11,10119	18,21	15,10	6000	1000	13104000
40320	0,1	8736	31584	16,76874	69,11	67,94	27000	4500	235872000
40320	0,8	8736	31584	44,40474	18,21	15,10	6000	1000	52416000
524160	0,1	113568	410592	217,99368	69,11	67,94	27000	4500	3066336000
524160	0,8	113568	410592	577,26168	18,21	15,10	6000	1000	681408000

Os valores de n usados representam uma hora ($n = 60$), um dia ($n = 1440$), uma semana ($n = 10080$), quatro semanas ($n = 40320$, aproximadamente um mês) e 52 semanas ($n = 524160$, um ano). O cenário de consumo de água, representado por w e z, foi ampliado na mesma escala dos incrementos de n, o que deve manter as estimativas de longo prazo consistentes. Podemos ver que a economia de energia (E e E_x) não é afetada por nenhuma outra variável que não seja x, graças à forma consistente com que alteramos w e z. Apenas a taxa de erro máxima do cenário (M_w) aumenta, o que é esperado dado o tempo extra considerado a cada incremento de n.

5.3 COMPARANDO DIFERENTES MODELOS DE SENSORES DE ÁGUA

Como o sensor de fluxo de água utilizado no protótipo tem uma alta taxa de erro de 25%, vamos compará-lo com duas outras opções: o YF-B7 (YF-B7, 2022) e o TUF-2000M (TUF-2000M, 2022). O TUF-2000M oferece mais recursos do que os modelos YF, mas também é proibitivamente caro para o escopo deste projeto, pois usa comprimentos de onda para medir o fluxo de água de um tubo. Já o YF-B7 é semelhante ao YFS-201b usado no protótipo, com algumas especificações diferentes e corpo metálico.

A Tabela 8 condensa todos os dados relevantes dos sensores para uma comparação rápida, e a Tabela 9 aplica dois cenários na Equação 10 para cada um dos três sensores. Os dados dos sensores foram obtidos de seus respectivos datasheets (YF-S201B, 2021; YF-B7, 2022; TUF-2000M, 2022). A vazão máxima que o TUF-2000M pode ler é para um tubo de 6000 mm de diâmetro com líquido fluindo a 7 m/s, e é por isso seus resultados são tão altos.

Tabela 8 – Dados relevantes de todos os três sensores de fluxo de água

Dispositivo	Por hora		Por minuto		Voltagem	Fluxo máximo	Preço
	mAh	Wh	mAh	Wh			
YF-S201b	15	0,075	0,25	0,00125	5V	30 l/min	R$48
YF-B7	15	0,075	0,25	0,00125	5V	25 l/min	R$86
TUF-2000M	41,67	1,5	0,6945	0,025	36V	4,28E+13 l/min	R$1800

A Tabela 9 mostra que os dois sensores YF consomem a mesma quantidade de energia em média, já que seus consumos nominais correspondem. O TUF-2000M, no entanto, consome muito mais energia, pois possui mais recursos e pode operar de forma independente. Os três sensores tiveram seu valor de M_w aumentado em 2300% ao incrementar n de 60 para 1440, o que significa que o aumento da quantidade de água que pode fluir não afeta a consistência da taxa máxima de erro. Os outros dois sensores não possuem valor M_{test} porque precisávamos tê-los em mãos para observar a vazão máxima de água que eles registrariam em nosso ambiente de testes.

Tabela 9 – Os valores estimados de Wh ao alterar todas as variáveis na Equação 10 para os três sensores diferentes

Dispositivo	n	x	w	z	Wh	$E\%$	$E_x(\%)$	$M_{spec}(ml)$	$M_{test}(ml)$	$M_w(ml)$
YFS-201b	60	0,5	13	47	0,04845	40,03	37,74	15000	2500	195000
YF-B7	60	0,5	13	47	0,04845	40,03	37,74	12500	N/A	162500
TUF-2000M	60	0,5	13	47	0,91533	-1032,91	-1076,08	2,14E+10	N/A	2,78E+11
YFS-201b	1440	0,5	312	1128	1,16288	40,03	37,74	15000	2500	4680000
YF-B7	1440	0,5	312	1128	1,16288	40,03	37,74	12500	N/A	3900000
TUF-2000M	1440	0,5	312	1128	21,96788	-1032,91	-1076,08	2,14E+10	N/A	6,68E+12

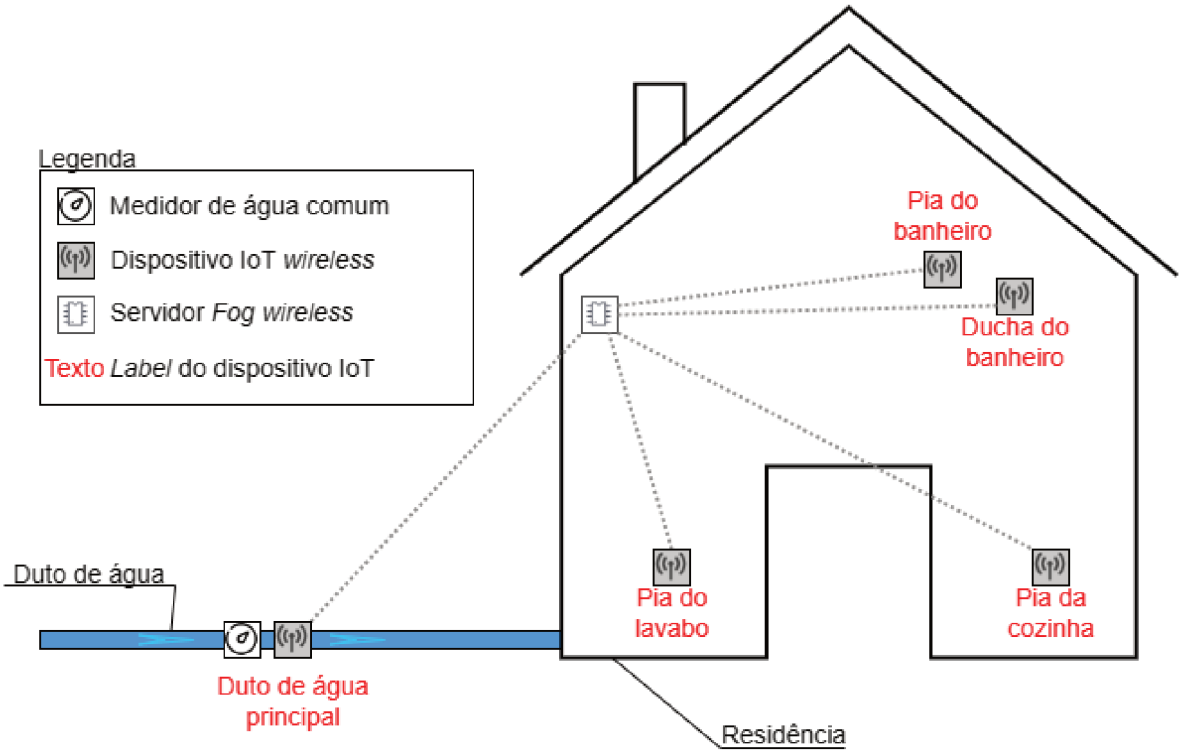
Embora possamos ver as semelhanças entre os dois sensores YF, o TUF-2000M se destaca por ser mais adequado para infraestrutura de cidade, pois podemos usá-lo em tubos muito maiores, de até 6 metros de diâmetro interno. Também podemos usá-lo para detectar enchentes, já que os canos de drenagem das cidades costumam entupir durante enchentes, o que o TUF-2000M pode identificar.

6 ANÁLISE DO SISTEMA DE MONITORAMENTO DE ÁGUA IOT EM LARGA ESCALA

Depois de mostrar como o protótipo pode ser usado, analisaremos a possibilidade de expandir a solução com vários nós IoT e para várias residências. Enquanto os testes do protótipo do dispositivo IoT foram feitos em um ambiente controlado, analisaremos esta nova solução multidispositivo do ponto de vista teórico, pois não podemos implementá-la agora.

6.1 USANDO MÚLTIPLOS SENSORES

Figura 14 – Visão geral da proposta com vários dispositivos IoT



Ao modificar a Figura 9, criamos a Figura 14, onde a residência agora abriga vários dispositivos IoT, que podemos usar para medir e rastrear o volume de água usado em vários pontos da casa ou apartamento. Esses dispositivos permitem maior granularidade nas funções da proposta do dispositivo, pois agora é possível detectar onde pode estar ocorrendo um vazamento e identificar onde a maior parte da água está sendo utilizada. Também podemos rastrear melhor o uso geral de água usando as leituras combinadas de todos os sensores dentro da casa para validar a leitura do sensor principal conectado ao tubo principal.

Criamos um cenário onde esses cinco dispositivos gerenciam o consumo de água em uma pequena casa onde moram duas pessoas. Usando a Equação 10 mais uma vez, vamos analisar um dia inteiro ($n = 1440$). Consideramos que ambos os moradores tomam um banho por dia, sendo que esse banho leva dez minutos, o que significa que temos um valor de $w = 20$

para o dispositivo de ducha do banheiro. Usando essa lógica, estimamos os números de uso de todos os dispositivos e criamos a Tabela 10.

Tabela 10 – O consumo estimado que cinco nós IoT gerariam na residência da Figura 14

Dispositivo	n	x	w	z	Consumo (em Wh)	Fluxo máximo (em l/min)	Volume máximo (em litros)
Duto de água principal	1440	0,5	52	1388	1,00038	60,5666	3149,46
Pia do lavabo	1440	0,5	2	1438	0,96913	8,32791	16,66
Pia da cozinha	1440	0,5	20	1420	0,98038	8,32791	166,56
Pia do banheiro	1440	0,5	10	1430	0,97413	8,32791	83,28
Ducha do banheiro	1440	0,5	20	1420	0,98038	9,46353	189,27

O valor w para o duto de água principal é a soma de todos os outros dispositivos IoT, pois este dispositivo principal detectará todo o fluxo de água que ocorrer. A Equação 10 fornece o consumo de energia de cada dispositivo por hora, que é afetado por x, o qual foi definido como 0,5 para mantê-lo "no meio". O volume máximo de água é medido pela multiplicação do fluxo máximo de cada dispositivo por w. Consideramos que o duto principal de água é um tubo de 19,05 *mm* (exatamente três quartos de polegada), permitindo um fluxo de água maior do que nosso protótipo consegue medir, fato que impede o cálculo das taxas de erro para este cenário, o que não é um problema, pois esta análise é teórica.

6.2 SOLUÇÃO DE LARGA ESCALA

Analisaremos agora uma implementação que abrange várias residências. A ideia é conectar todas as residências de um condomínio em um sistema que coleta dados de vários usuários e os notifica individualmente, além de permitir que o síndico fique de olho em todos.

Em nosso cenário, cada residência terá cinco dispositivos IoT (como os representados na Figura 14) e um servidor *Fog*. Vamos considerar um condomínio muito denso com 300 apartamentos, todos no mesmo sistema. Por fim, um servidor *Fog* extra será considerado o *hub* principal que processa os dados de todas as residências.

Quanto aos valores de uso, criamos três tipos de apartamentos. A Tabela 11 mostra os valores w dos sensores para cada tipo de apartamento e, para as próximas tabelas, dividimos esses tipos da seguinte forma:

- 10% são do tipo 1 (ativo), que representa o tipo de apartamento onde os moradores estão o dia todo em casa e usam água com mais frequência que a maioria das pessoas, resultando em valores maiores de w;
- 75% são do tipo 2 (normal), que representa o tipo de apartamento onde os moradores fazem um uso mais normal da água, com valores de w medianos; e
- 15% são do tipo 3 (inativo), que representa o tipo de apartamento onde os moradores passam o dia todo fora, resultando em valores de w pequenos.

Tabela 11 – Os três tipos de apartamentos

Tipo de apartamento	*w* do duto de água principal	*w* da pia do lavabo	*w* da pia da cozinha	*w* da pia do banheiro	*w* da ducha do banheiro
Tipo 1 (ativo)	130	10	60	20	40
Tipo 2 (normal)	52	2	20	10	20
Tipo 3 (inativo)	10	0	2	3	5

A Tabela 12 mostra o consumo médio de energia de cada tipo de apartamento e a média dos três tipos somados. Fizemos cálculos usando a Equação 10 para estimar o consumo para os mesmos quatro períodos vistos na Tabela 7 (exceto para o período de uma única hora): um dia (n = 1440), uma semana (n = 10080), quatro semanas (n = 40320, aproximadamente um mês) e 52 semanas (n = 524160, um ano). Mantendo o valor de x consistente e aumentando w e z na mesma escala de n, o consumo também aumenta de acordo com a escala. Esse comportamento é mais fácil de observar analisando o valor de w dos apartamentos tipo 3 para cada n.

Tabela 12 – O consumo de energia estimado que a solução geraria em 300 apartamentos em diferentes períodos

Dispositivo	Qtd	n	x	w	z	Consumo (em Wh)
Duto de água do condomínio	1	1440	0,5	221	1219	1,1060
Apartamentos do tipo 1	30			130	1310	150,0576
Apartamentos do tipo 2	225			52	1388	1103,4942
Apartamentos do tipo 3	45			10	1430	218,3363
Todos os apartamentos	300			192	1248	1471,8881
Duto de água do condomínio	1	10080	0,5	1547	8533	7,7421
Apartamentos do tipo 1	30			910	9170	1050,4029
Apartamentos do tipo 2	225			364	9716	7724,4594
Apartamentos do tipo 3	45			70	10010	1528,3544
Todos os apartamentos	300			1344	8736	10303,2167
Duto de água do condomínio	1	40320	0,5	6188	34132	30,9682
Apartamentos do tipo 1	30			3640	36680	4201,6117
Apartamentos do tipo 2	225			1456	38864	30897,8377
Apartamentos do tipo 3	45			280	40040	6113,4175
Todos os apartamentos	300			5376	34944	41212,8669
Duto de água do condomínio	1	524160	0,5	80444	443716	402,5872
Apartamentos do tipo 1	30			47320	476840	54620,9520
Apartamentos do tipo 2	225			18928	505232	401671,8899
Apartamentos do tipo 3	45			3640	520520	79474,4280
Todos os apartamentos	300			69888	454272	535767,2699

Se pegarmos os valores w da Tabela 11 e os colocarmos na Tabela 12, podemos somar o consumo do duto de água do condomínio e de todos os apartamentos para estimar que este sistema deve consumir cerca de 1472 Wh diariamente. Como não podemos garantir que os apartamentos irão ligar seus sensores ao mesmo tempo, calculamos o valor de w para o duto de água do condomínio de forma diferente. Ao invés de simplesmente ser igual ao maior valor de qualquer tipo de apartamento, é 15% maior do que todos os tipos somados (por exemplo, $192 + 15\% \cong 221$). Essa porcentagem dá uma margem extra para o cenário simulado.

6.3 ESTUDO DO IMPACTO DA DENSIDADE EM UM SISTEMA DE ÁGUA DE LARGA ESCALA

Neste trabalho, verificamos uma forma de gerenciar a energia para obter economia extra de energia. Também verificamos como este sistema de água se comportaria numa implementação de larga escala, porém, tal implementação não traz apenas problemas de consumo de energia. Se faz necessária uma análise da performance da rede que conecta todos os dispositivos IoT utilizados em sua implementação.

No caso do cenário explorado, os apartamentos podem estar distribuídos de diversas maneiras, o que significa que o posicionamento das antenas dos dispositivos IoT pode afetar a qualidade da rede como um todo. Isso se deve ao problema da densidade de antenas e, subsequentemente, de pacotes de rede. Uma quantidade muito elevada de pacotes sendo transmitidos de maneira *wireless* apresenta um risco elevado de colisões e perdas desses pacotes, o que pode afetar o funcionamento do sistema, causando *delays* ou até mesmo perda de dados. O reposicionamento estratégico das antenas e o ajuste da taxa de envio de pacotes por antena podem ajudar a mitigar o problema de densidade.

Será feita uma análise de uma rede de larga escala de forma teórica e simulada. A teoria de redes de filas será utilizada na análise teórica, enquanto um simulador de redes, especificamente o NS-3, será utilizado na análise simulada.

6.3.1 Análise teórica

A análise teórica utilizará a teoria de redes de filas (apresentada na Seção 2.2) para analisar o impacto da densidade de pacotes em uma rede de larga escala (como a analisada no Capítulo 6). As Equações 11 e 12 nos permitem encontrar as variáveis necessárias para resolver a Equação 13, a qual foi adaptada da Seção 2.2 e de Sampaio et al. (2021). Já a Equação 14 nos permite encontrar a variável necessária para resolver a Equação 15, que é a equação principal desta análise teórica.

$$\mu = \frac{B}{S} \tag{11}$$

As variáveis B e S da Equação 11 representam o *baud rate* da porta serial do nó (medido em bits por segundo (bps)) e o tamanho (em bits) de cada pacote sendo recebido pelo nó, respectivamente. O resultado μ representa a taxa de saída de pacotes do nó IoT por segundo.

$$\lambda = \frac{Q}{T} \tag{12}$$

As variáveis Q e T da Equação 12 representam a quantidade de pacotes sendo enviados na rede e o tempo (em segundos) entre o envio de cada um desses pacotes, respectivamente. O valor de Q tende a ser igual à quantidade de nós na rede, enquanto que o valor de T pode ser definido como a quantidade de tempo que os nós esperam antes de enviar seu próximo pacote

(o "*delay*" de envio). Por exemplo, se temos 100 nós enviando um pacote a cada 30 segundos, temos $Q = 100$ e $T = 30$, enquanto que 50 nós enviando 2 pacotes por segundo nos daria $Q = 100$ e $T = 1$. O resultado λ representa a taxa de entrada de pacotes no nó IoT por segundo.

$$E\{T\} = \frac{1}{\mu - \lambda} \tag{13}$$

A Equação 13 (réplica da Equação 1 dos conceitos fundamentais) resulta em $E\{T\}$, que representa o tempo médio que 1 pacote gasta em cada fila de espera de pacotes, e é isto que causa o gargalo de processamento na porta serial. O resultado da equação é um valor médio pois, para saber o tempo total que levará para um pacote ser processado, seria necessário saber quanto tempo todos os pacotes na frente da fila levarão para serem processados, além de também ser necessário calcular quanto tempo ainda falta para processar o pacote que já está sendo processado. Como as contas necessárias geram uma complexidade extra de maneira desnecessária, a Equação 13 produz um resultado mais simples, um tempo médio.

$$D = \frac{\lambda \times S}{C} \text{ , } onde\ C = B \times \log_2(1 + \frac{P}{N}) \tag{14}$$

O objetivo da Equação 14 é calcular o *delay* de envio de pacotes na rede, a variável D (medida em segundos). Essa equação utiliza a fórmula do teorema de Shannon. Dividimos o tamanho dos pacotes sendo enviados ($\lambda \times S$) pela capacidade da rede (C, obtido da Equação 3), o que resulta na quantidade de tempo que um pacote leva para "percorrer" a rede até chegar no nó final. Este *delay* D é causado tanto pelo tipo da rede, ou seja, toda as variáveis do teorema de Shannon, quanto pela quantidade de antenas, que podem afetar o valor de *signal-to-noise ratio* ($\frac{P}{N}$), já que as mesmas geram ruído na rede durante o seu funcionamento. O valor de D é considerado uma estimativa, pois tanto a Equação 14 quanto a fórmula de Shannon (utilizada na equação) não levam em consideração as colisões de pacotes, fator que pode aumentar o valor de D na prática.

$$DT = E\{T\} + D \tag{15}$$

Por fim, a Equação 15 resulta em DT, o valor que representa o "*delay* total" da rede.

6.3.1.1 Cálculos considerando 300 apartamentos (nós)

Para esta análise, utilizaremos todas as fórmulas apresentadas para achar o *delay* total do sistema considerado na rede de larga escala do Capítulo 6, o qual apresenta 300 nós IoT, um por apartamento. Os outros valores relevantes são obtidos do protótipo desenvolvido no Capítulo 4, porém não diretamente de dito Capítulo (nem todos os valores do protótipo foram apresentados neste trabalho). Temos os valores:

- $B = 9600\ bps$ (o *baud rate* utilizado pelo protótipo)

- $S = 20\ bytes = 160\ bits$ (o maior pacote produzido pelo protótipo era de 19 bytes, então usamos 20 para facilitar as contas)
- $Q = 300\ pacotes$ (a quantidade de pacotes é idêntica a de nós, pois apenas um pacote é enviado por nó a cada T segundos)
- $T = 60\ segundos$ (o intervalo padrão de envio de pacotes do protótipo)

Aplicando os valores apresentados nas Equações 11, 12 e 13, temos, respectivamente, as Equações 16, 17 e 18.

$$\mu = \frac{9600}{160} = 60\ pacotes\ por\ segundo \tag{16}$$

$$\lambda = \frac{300}{60} = 5\ pacotes\ por\ segundo \tag{17}$$

$$E\{T\} = \frac{1}{60-5} = \frac{1}{55} = 0,01818\ segundos \tag{18}$$

O protótipo desenvolvido atualiza a leitura de água, por padrão, a cada 60 segundos, ou a cada minuto. A quantidade de pacotes enviada na rede a qualquer momento é pequena, apenas cinco pacotes por segundo de acordo com a Equação 17. Como a porta serial do nó que recebe os pacotes é capaz de processar 60 pacotes (com tamanho de 160 bits cada) por segundo (Equação 16), não existe um gargalo neste sistema nesta escala. Cada pacote fica, em média, 18,1 milissegundos na fila, de acordo com a Equação 18. Os 5 pacotes que chegam a cada segundo levam um total de 90,5 milissegundos para serem processados.

Consideramos agora que a rede *wireless* opera usando WiFi, especificamente 802.11n. Neste caso, teremos os valores de largura de banda $B = 20MHz$, *signal-to-noise ratio* $\frac{P}{N} = 20$ (em geral, o valor considerado mínimo para uma boa conexão) e tamanho dos pacotes $S = 0,00016Mbits$ (convertido de *bits*). Por fim, também consideramos que o *delay* D é amplificado pela quantidade de pacotes que pode ser enviada simultaneamente a cada segundo, que neste caso seria cinco, pois $\lambda = 5$. Aplicamos estes valores na Equação 14, com os resultados apresentados na Equação 19.

$$D = \frac{5 \times 0,00016}{20 \times \log_2(1+20)} = \frac{0,0008}{20 \times 4,39232} = \frac{0,0008}{87,84635} = 0,0000091\ segundos \tag{19}$$

Este resultado demonstra que a rede sendo considerada, por si só, causa um *delay* de 0,0000091 s para cada λ pacotes. Podemos somar este valor a $E\{T\}$ para obter o valor do *delay* total DT:

$$DT = 0,01818 + 0,0000091 = 0,0181891\ segundos \tag{20}$$

Utilizando a Equação 15, temos a Equação 20, que mostra que $DT = 0,0181891$, ou seja, devido à densidade de 300 nós na rede, o *delay* total de um pacote é, em média, 18 milissegundos.

6.3.2 Análise simulada

Esta análise será uma evolução do trabalho que foi feito em Sampaio et al. (2021), onde o atraso extra causado pela densidade da rede foi medido considerando apenas o momento em que um pacote chega na *Fog*, ou seja, o atraso de tempo do processamento dos dados na porta serial do dispositivo que os recebe. Passaremos a também considerar o tempo de atraso antes do pacote chegar à *Fog*, o que inclui o tempo perdido em retransmissão e perda de pacotes, que é justamente o atraso mais influenciado pela densidade de pacotes sendo enviados na rede. Na simulação, os problemas causados pela densidade, como erros de transmissão, interferência, perda de pacotes, dentre outros, serão analisados.

Os dois objetivos principais são calcular a *DT*, ou seja, resolver a Equação 15, e apresentar uma análise completa de como alterações na densidade de uma rede, seja por redução da quantidade de pacotes sendo enviados simultaneamente ou pela alteração do posicionamento de antenas, podem afetar a sua performance e confiabilidade.

6.3.2.1 Network Simulator 3 (NS-3)

O *Network Simulator 3* (NS-3) é um simulador de redes (de eventos discretos) gratuito e *open source*. Ele foi projetado para pesquisa e educação em redes, fornecendo uma plataforma realista e flexível para modelagem e simulação de vários protocolos, arquiteturas e cenários de rede. O NS-3 é mantido por uma comunidade mundial de desenvolvedores e usuários e é licenciado sob a licença GNU GPLv2.

O NS-3 pode ser baixado de seu site oficial, onde os usuários também podem encontrar documentação, tutoriais, exemplos e outros recursos. O NS-3 pode ser usado para várias finalidades, como avaliar o desempenho e estudar o comportamento de redes, e também para testar novos algoritmos de rede. O NS-3 suporta muitas tecnologias de rede, como WiFi, LTE, TCP/IP, protocolos de roteamento, modelos de mobilidade, etc.

6.3.2.1.1 Simulações iniciais

Alguns experimentos iniciais foram conduzidos para testar o NS-3. O experimento mais robusto é o `experiment_v6.cc`, que pode ser observado no apêndice A. Este código C++ configura uma simulação de rede sem fio com vários pontos de acesso (`AP`) e nós de estação (`STA`). A simulação envolve a criação de vários nós `AP` com um número fixo de nós `STA` por `AP` e a configuração de uma rede sem fio com esses nós. Os nós recebem o atributo *mobility* usando um modelo de mobilidade de posição constante e o *internet stack* é configurado para os nós. Os endereços IP são atribuídos aos nós e, finalmente, aplicativos UDP *Echo Server* e *Echo Client* são instalados nos nós para teste.

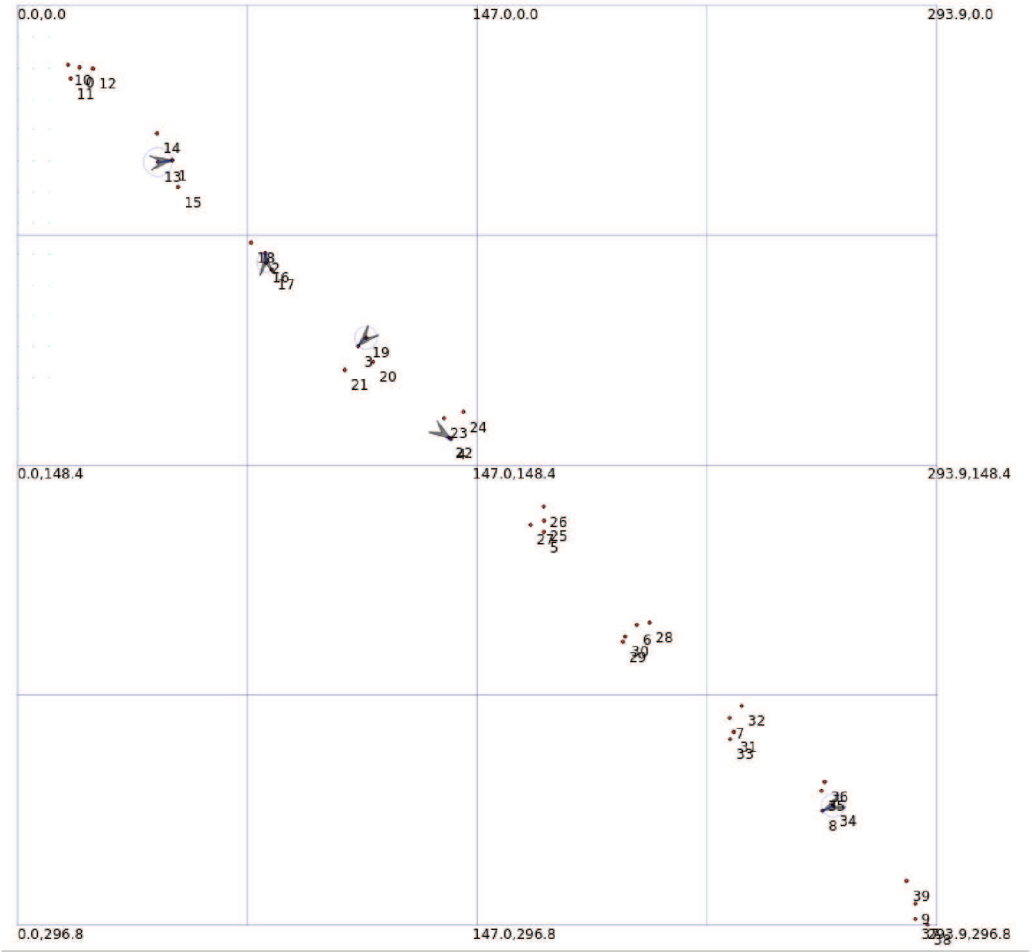
O código começa com variáveis e *booleans* para registro e rastreamento (*logging* e *tracing*). Em seguida, um contêiner de nós é criado para os nós `AP` e `STA`, onde o número de nós

é especificado pelas variáveis `apNodeCount` e `staNodeCount`. O `YansWifiChannelHelper`, `YansWifiPhyHelper`, `WifiHelper`, `WifiMacHelper`, `MobilityHelper`, `InternetStackHelper` e `Ipv4AddressHelper` são inicializados e o método `wifi.SetRemoteStationManager()` é usado para especificar o gerenciador de estação usado pela rede sem fio, o AARF (*Adaptive Advanced Routing and Forwarding*, ou Encaminhamento e Roteamento Avançado Adaptável).

O código então cria *NetDeviceContainers* e instala esses dispositivos nos nós `AP` e `STA` da rede utilizando um *loop* que cria um SSID para cada nó, atualiza o valor da variável `phy` para evitar confusão com os números da rede, e atualiza o `mac` para cada tipo de nó. O modelo de mobilidade e o alocador de posição são definidos para os nós e o *internet stack* é instalado. Os endereços IP são atribuídos aos nós usando o `Ipv4InterfaceContainer` e um *loop* que cria ditos endereços por partes antes da atribuição.

Por fim, os aplicativos UDP são criados sem o uso de *loops*, pois não existe um construtor *default*. O código cria um servidor em cada nó `AP` usando a classe `UdpEchoServerHelper` e define o número da porta. Em seguida, o código cria um cliente em cada nó `STA` usando a classe `UdpEchoClientHelper` e define o endereço do servidor de acordo com o endereço do nó `AP` correspondente, além de também definir a porta com o mesmo número utilizado no passo anterior. A classe `ApplicationContainer` é usada para armazenar os aplicativos de servidor e cliente.

Figura 15 – Exemplo de animação no NetAnim, resultado do `experiment_v6`



A última parte do código cria arquivos PCAP e um arquivo de animação da simulação. Os arquivos PCAP contém toda a atividade de comunicação da rede e podem ser analisados utilizando o programa WireShark. O arquivo de animação cria uma representação visual da rede, que pode ser analisada com uma ferramenta do NS-3 chamada NetAnim, como demonstrado na Figura 15. É importante destacar que as coordenadas dos nós definidas para criar essa animação também influenciam a simulação diretamente, ou seja, caso os nós sejam posicionados muito distantes na animação, eles não se comunicarão na simulação.

6.4 TRABALHOS FUTUROS

Considerando o protótipo e sistema desenvolvidos, pretendemos continuar desenvolvendo o protótipo para medir uma quantidade mais variada de dados, como pressão, temperatura, pureza e pH da água. Esse desenvolvimento também deve focar no teste no mundo real de vários sensores usados para monitorar várias saídas de água de uma casa, permitindo a medição de torneiras, chuveiros e mangueiras específicas individualmente. Também é necessária uma pesquisa mais aprofundada sobre uma alternativa adequada ao modelo de sensor utilizado, preferencialmente para um modelo com uma taxa de erro muito menor para tornar o protótipo mais utilizável em cenários do mundo real.

Considerando o estudo do impacto da densidade em um sistema de monitoramento de água em larga escala, apresentamos quatro direções possíveis para expandir o estudo.

Uma das direções possíveis é continuar a pesquisa considerando *clusters* de nós. *Clusters* são grupos de dispositivos que se comunicam entre si e compartilham dados ou recursos. O *clustering* pode melhorar a escalabilidade, a confiabilidade e a eficiência das redes IoT, reduzindo a sobrecarga e o congestionamento. No entanto, o *clustering* também apresenta desafios, como formar e manter *clusters*, equilibrar a carga entre os membros do *cluster*, lidar com a mobilidade e a heterogeneidade dos dispositivos e garantir a segurança e a privacidade nos *clusters*.

Outra direção possível é continuar a pesquisa considerando a quantidade de dispositivos. Espera-se que o número de dispositivos IoT cresça exponencialmente, atingindo bilhões ou até trilhões, e isso criará demandas sem precedentes por largura de banda, energia, armazenamento e poder de processamento. Será importante investigar como otimizar o desempenho das redes IoT considerando diferentes densidades de dispositivos e cenários de distribuição. Além disso, também é importante explorar como lidar com a dinamicidade e a imprevisibilidade do comportamento do dispositivo e dos padrões de tráfego.

Uma terceira direção possível é continuar a pesquisa considerando diferentes protocolos. Os protocolos controlam como os dispositivos se comunicam entre si e com outras entidades, como servidores ou *gateways*. É importante estudar como diferentes protocolos afetam o desempenho das redes IoT em vários aspectos, como *throughput*, *delay*, taxa de perda de pacotes, eficiência energética, segurança, confiabilidade, velocidade e alcance. Alguns exemplos de protocolo são a LoRaWAN, que é um protocolo de rede de área ampla de baixa potência

(*low-power wide-area network protocol*) que permite comunicação de longo alcance com baixas taxas de dados; Zigbee, que é um protocolo sem fio de curto alcance que suporta redes *mesh* e altas taxas de dados; 5G, que é um protocolo de rede celular que oferece conectividade de banda larga de alta velocidade com baixa latência; e WiFi (variantes de 2,4 e 5 GHz), que são protocolos de rede local sem fio que fornecem transmissão rápida de dados em distâncias curtas.

Por fim, uma quarta direção possível é a implementação de um algoritmo de transmissão de pacotes, que será apresentado na Subseção 6.4.1. Os testes desse algoritmo seriam conduzidos em um ambiente simulado, pois o grande foco são as redes de larga escala, e montar uma dessas redes apenas para experimentos seria muito caro e um desperdício de tempo.

6.4.1 Algoritmo de transmissão de pacotes

Os protocolos tradicionais TDMA (*Time Division Multiple Access*) e CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*) tornaram-se menos eficientes em lidar com o aumento do tráfego trazido por redes IoT de larga escala, seja pela limitação temporal do TDMA ou por *overhead* de pacotes de controle em protocolos CSMA (de acordo com Kosunalp e Kaya (2022)). Para contornar esse problema, propomos um novo algoritmo que estende os protocolos TDMA e CSMA/CA.

6.4.1.1 As técnicas existentes

De acordo com A… (2020), a técnica utilizada pelo TDMA divide o tempo total de duração da comunicação em um número fixo de intervalos de tempo que são configurados em quadros de tempo que se repetem periodicamente (os *time slots*). O problema de utilizar TDMA em redes IoT de larga escala é a necessidade de sincronização entre os dispositivos que enviam pacotes, o que pode tornar-se muito custoso, especialmente em redes com tamanho dinâmico. Um outro problema do TDMA é o desperdício de banda por dispositivos que não tem nada a transmitir durante o seu *time slot*.

De acordo com Peng e Cheng (2006), o CSMA/CA é um protocolo MAC usado em redes sem fio. Com CSMA/CA, o remetente não transmite imediatamente o pacote de dados após atrasos e recuos adequados (ou seja, após reivindicar com sucesso o meio de comunicação com CSMA). Em vez disso, ele envia um RTS (*Request To Send*) para contatar o destinatário e reservar o meio de comunicação para si. Depois de receber o quadro RTS, o destinatário envia um CTS (*Clear To Send*) para responder ao remetente e reservar o meio de comunicação também. Se o processo de *handshake* e reserva do meio for bem-sucedido, o remetente começa a transmitir o pacote de dados. Quando um vizinho recebe um quadro RTS ou CTS, ele verifica o tempo de duração do quadro para saber por quanto tempo deve recuar. Um terminal oculto do remetente será, portanto, suprimido pelo quadro CTS gerado pelo destinatário pretendido se o terminal oculto receber com sucesso o quadro CTS.

Todos os pacotes de *overhead* utilizados pelo CSMA/CA reduzem a eficiência de uso do canal de comunicação e aumentam o consumo de energia dos dispositivos IoT. O CSMA/CA também sofre com o problema do nó oculto (*Hidden Node problem*), pois pode ocorrer que dois nós que estão ao alcance do nó receptor não estão ao alcance um do outro, o que pode resultar em muitas colisões consecutivas. Por fim, o CSMA/CA também não oferece garantias de QoS para os vários tipos de redes IoT, como as de *real time* e as *delay sensitive*.

Os pontos positivos dos dois protocolos descritos incluem uma divisão temporal para o envio de pacotes, vinda do TDMA, e uma tentativa de evitar colisões o máximo possível, vinda do CSMA/CA.

6.4.1.2 A proposta

O algoritmo proposto pretende separar o envio de pacotes em "*pseudo-time slots*", ou seja, os pacotes serão enviados em *slots* de tempo separados, porém tais *slots* não existirão na prática. A separação de tais *slots* será dada pela variável DT (Subseção 6.3.1).

Ao inicializar a rede IoT de larga escala, durante a fase que chamaremos de *setup*, o nó receptor ordenará cada dispositivo a iniciar a sua função. Essa ordenação será feita de acordo com o *ping* de cada nó em relação ao nó receptor. A inicialização de cada nó sensor será feita com um *delay* DT entre cada, de forma a que todos os nós operem em tempos levemente diferentes uns dos outros.

No caso do sistema de monitoramento de água, como todos os sensores enviam uma nova leitura a cada x segundos, o nó receptor receberia todos os pacotes de uma só vez. Porém, se cada nó estiver operando com um *delay* em relação a todos os outros nós, os pacotes chegarão ao nó receptor no momento em que o pacote anterior já foi processado. É importante que tais *delays* não superem x, senão, os últimos sensores da lista ficarão perceptivelmente atrasados em relação ao resto do sistema ($DT \times$ quantidade de nós $\leq x$). O limite na quantidade de *delay* que pode ser utilizado indiretamente significa que a quantidade máxima de sensores que a estratégia poderá acomodar é influenciada por x.

Esta estratégia evita a necessidade de sincronização entre os nós e evita as colisões de pacotes proativamente, aproveitando os aspectos positivos da TDMA e da CSMA/CA.

Além da análise de performance da rede para verificar se esta solução é capaz de melhorar a eficiência na comunicação dos nós de forma significativa, também será feita uma nova análise de consumo de energia, para verificar como este novo algoritmo afetará os gastos do sistema IoT a longo prazo.

7 CONCLUSÃO

Considerando o paradigma de energia IoT, implementamos uma solução para o sistema de gerenciamento autônomo de consumo de água de uma residência. Desenvolvemos um protótipo de IoT para obter medições de consumo de água enquanto controlamos o ciclo de trabalho da IoT para obter economia de energia.

Propusemos alterar o uso do sensor de fluxo de água considerando dois estados de funcionamento, com ou sem fluxo de água, desligando o sensor de água quando não houver fluxo de água e ligando-o constantemente para verificar se há fluxo de água. Embora o controle da antena oferecesse alguma economia de energia, ter alguma maneira de reduzir o consumo do sensor de fluxo de água pode ter um impacto dramático no consumo geral de energia da IoT ou na longevidade da bateria.

Nossos resultados mostraram que poderíamos obter cerca de 69% de economia de energia extra em comparação com apenas colocar a antena para dormir. Há um *trade-off* observável em economizar tanta energia, pois também podemos ver que as taxas de erro de leitura de água aumentam junto com a economia extra de energia. Embora nossos testes de energia tenham sido divididos entre energia da bateria e linha de energia, o uso de baterias em um ambiente doméstico inteligente geralmente não é o ideal. Fizemos os testes com baterias porque esse sistema pode ser implantado em vários ambientes, como casas rurais, onde o uso de baterias pode fazer mais sentido.

Também analisamos como podemos expandir essa solução para sistemas maiores e como o consumo de energia se comportaria em tal ambiente. Essa análise inclui uma residência multissensor e uma operação em larga escala para todo um condomínio, e leva em consideração os padrões de consumo de água, separando os apartamentos em três tipologias.

Por fim, propusemos um estudo com foco na performance da rede dos dispositivos IoT utilizados na solução de larga escala, tendo em vista que a densidade de dispositivos e a quantidade de pacotes enviados na rede podem causar problemas de estabilidade e de funcionamento da solução.

REFERÊNCIAS

A survey on time division multiple access scheduling algorithms for industrial networks. *SN Applied Sciences*, v. 2, 2020. ISSN 25233971.

AGGARWAL, S.; CHAUHAN, S.; PRAKASH, R. J. An Automated System to Monitor the Usage of Water in Apartments Using IOT and Artificial Neural Network. In: **2019 2nd International Conference on Intelligent Computing, Instrumentation and Control Technologies (ICICICT)**. [S.l.]: IEEE, 2019. p. 821–825. ISBN 978-1-7281-0282-5.

ALRAWAIS, A. et al. Fog Computing for the Internet of Things: Security and Privacy Issues. **IEEE Internet Computing**, IEEE, v. 21, n. 2, p. 34–42, 2017.

ARDUINO. Arduino Reference Manual. 2021. Acessado em Maio de 2021. Disponível em: https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf.

BHILARE, R.; MALI, S. IoT based smart home with real time E-metering using E-controller. **12th IEEE International Conference Electronics, Energy, Environment, Communication, Computer, Control: (E3-C3), INDICON 2015**, p. 1–6, 2016.

CôRTE, P. et al. Iot energy management for smart homes' water management system. **Journal of Circuits, Systems and Computers**, v. 1, n. 1, p. 25, 2023.

ESP8266, A. ESP8266 Specifications English. 2021. Acessado em Junho de 2021. Disponível em: https://cdn-shop.adafruit.com/datasheets/ESP8266_Specifications_English.pdf.

IORGA, M. et al. Fog computing conceptual model. **NIST Special Publication**, n. 500-325, p. 11, 2018.

JEURKAR, V. et al. IOT Based Water Management System. **2020 International Conference on Industry 4.0 Technology, I4Tech 2020**, p. 141–144, 2020.

KODALI, R. K.; KIRTI, B. NS-3 Model of an IoT network. **2020 IEEE 5th International Conference on Computing Communication and Automation, ICCCA 2020**, p. 699–702, 2020.

KOSUNALP, S.; KAYA, Y. Iot-tdma: A performance evaluation of tdma scheme for wireless sensor networks with internet of things. **Concurrency and Computation: Practice and Experience**, John Wiley and Sons Ltd, v. 34, 9 2022. ISSN 15320634.

MA, J. Modified shannon's capacity for wireless communication [speaker's corner]. **IEEE Microwave Magazine**, Institute of Electrical and Electronics Engineers Inc., p. 97–100, 2021. ISSN 15579581.

NARAYANAN, L. K.; SANKARANARAYANAN, S. IoT Enabled Smart Water Distribution and Underground Pipe Health Monitoring Architecture for Smart Cities. In: **2019 IEEE 5th International Conference for Convergence in Technology (I2CT)**. [S.l.]: IEEE, 2019. p. 1–7. ISBN 978-1-5386-8075-9.

OUKESSOU, Y.; BASLAM, M.; OUKESSOU, M. Lpwan ieee 802.11ah and lorawan capacity simulation analysis comparison using ns-3. In: . [S.l.]: Institute of Electrical and Electronics Engineers Inc., 2018. p. 1–4. ISBN 9781538642252.

PASSOS, D. et al. Asynchronous Radio Duty Cycling for Green IoT: State of the Art and Future Perspectives. **IEEE Communications Magazine**, v. 57, n. 9, p. 106–111, 2019. ISSN 15581896.

PATEL, H. K.; MODY, T.; GOYAL, A. Arduino Based Smart Energy Meter using GSM. **Proceedings - 2019 4th International Conference on Internet of Things: Smart Innovation and Usages, IoT-SIU 2019**, p. 1–6, 2019.

PENG, J.; CHENG, L. Revisiting carrier sense multiple access with collision avoidance (csma/ca). **2006 40th Annual Conference on Information Sciences and Systems**, p. 1236–1241, 3 2006.

PETERSON, L. L.; DAVIE, B. S. Computer networks. Morgan Kaufmann, 4 1996.

RAY, A.; RAY, H. SSIWM: Smart Secured IoT Framework for Integrated Water Resource Management System. In: **2020 IEEE Bangalore Humanitarian Technology Conference (B-HTC)**. [S.l.]: IEEE, 2020. p. 1–6. ISBN 978-1-7281-8794-5.

SAMPAIO, H.; MOTOYAMA, S. Sensor nodes estimation for a greenhouse monitoring system using hierarchical wireless network. **2017 25th International Conference on Software, Telecommunications and Computer Networks, SoftCOM 2017**, 2017.

SAMPAIO, H. V. et al. Autonomic IoT Battery Management with Fog Computing. **Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)**, v. 11484 LNCS, n. Cc, p. 89–103, 2019.

SAMPAIO, H. V. et al. Autonomic energy management with Fog Computing. **Computers and Electrical Engineering**, v. 93, n. May, p. 1–15, 2021. ISSN 00457906.

SHANNON, C. E. A mathematical theory of communication. **The Bell System Technical Journal**, v. 27, n. 3, p. 379–423, 1948.

TUF-2000M, S. T. TUF-2000M datasheet. 2022. Acessado em Dezembro de 2022. Disponível em: https://5.imimg.com/data5/AP/ML/GH/SELLER-72826034/ultrasonic-btu-meter.pdf.

YF-B7, S. studio. YF-B7 Specification English. 2022. Acessado em Dezembro de 2022. Disponível em: https://media.digikey.com/pdf/Data

YF-S201B, A. YF-S201 Specification English. 2021. Acessado em Maio de 2021. Disponível em: https://cdn-shop.adafruit.com/product-files/828/C898+datasheet.pdf.

ZUKERMAN, M. Introduction to queueing theory and stochastic teletraffic models. arXiv, 2022. Disponível em: https://arxiv.org/abs/1307.2968v25.

A ANEXO A

Algumas alterações foram feitas no código para melhorar a legibilidade neste PDF. Este código pode ser visualizado em:

https://github.com/pedroabcorte/codigoAnexoADissertacaoPedroAlexandre2023.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/mobility-module.h"
#include "ns3/internet-module.h"
#include "ns3/yans-wifi-helper.h"
#include "ns3/ssid.h"
#include "ns3/netanim-module.h"
#include "ns3/flow-monitor-module.h"
#include <random>

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("experiment_v6");

int
main (int argc, char *argv[])
{
  bool verbose = true; // Decide whether logging info should be printed in the Terminal or not
  bool tracing = true; // Decide whether simulation traces should be saved to files or not
  double globalStopTime = 10; // The stop time used everywhere

  uint32_t apNodeCount = 10; // Number of Access Point nodes
  uint32_t staNodeCount = 3; // Number of Station per AP node

  if (verbose)
    {
      // Log UDP Echo Client data in the Terminal
      LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);

      // Log UDP Echo Server data in the Terminal
      LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
    }

  // Create the container for all of the nodes
  NodeContainer apNodes;
  NodeContainer staNodes[apNodeCount];

  apNodes.Create (apNodeCount);
  for (uint32_t i = 0; i < apNodeCount; i++)
    {
```

```
42        staNodes[i].Create (staNodeCount);
43      }
44
45    // Create all the helpers needed for networking (they won't be used
          right away)
46    YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
          // Channel
47    YansWifiPhyHelper phy = YansWifiPhyHelper::Default (); // Physical
          layer
48    WifiHelper wifi; // General WiFi helper
49    WifiMacHelper mac; // MAC protocol
50    /* The SSID is not created here, as it is not necessary */
51    MobilityHelper mobility; // Generic Mobility (needed for wireless
          networks)
52    InternetStackHelper stack; // Internet stack helper
53    Ipv4AddressHelper address; // IP address helper
54
55    /// Set up the basic parameters for PHY and WiFi
56    wifi.SetRemoteStationManager ("ns3::AarfWifiManager"); // Adaptive
          Advanced Routing and Forwarding
57
58    // Create the container for all net devices (part of the nodes)
59    NetDeviceContainer apNetDevices[apNodeCount];
60    NetDeviceContainer staNetDevices[apNodeCount];
61
62    // Install the devices in the nodes within the network
63    for (uint32_t i = 0; i < apNodeCount; i++)
64      {
65        /// Create the SSID for this loop
66        std::ostringstream oss;
67        oss << "SSiD-" << i;
68        Ssid ssid = Ssid (oss.str ());
69
70        /// Update the phy channel to avoid Network Number Confusion
71        phy.SetChannel (channel.Create ()); // Set the PHY channel
              while creating it
72
73        /// Set one AP node to currentSsid
74        mac.SetType ("ns3::ApWifiMac", "Ssid", SsidValue (ssid));
75        apNetDevices[i] = wifi.Install (phy, mac, apNodes.Get (i));
76
77        /// Set its Station nodes to the same currentSsid
78        mac.SetType ("ns3::StaWifiMac", "Ssid", SsidValue (ssid), "
              ActiveProbing",
79                     BooleanValue (false));
80        staNetDevices[i] = wifi.Install (phy, mac, staNodes[i]);
81      }
82
83    // Install the mobility aspect of the nodes
84    mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
85    mobility.SetPositionAllocator ("ns3::GridPositionAllocator", "Z",
```

```
      DoubleValue (1), "MinX", DoubleValue (0.0), "MinY", DoubleValue
       (0.0), "DeltaX", DoubleValue (5.0), "DeltaY", DoubleValue
      (10.0), "GridWidth", UintegerValue (3), "LayoutType",
      StringValue ("RowFirst"));
  mobility.Install (apNodes);
  for (uint32_t i = 0; i < apNodeCount; i++)
    {
      mobility.Install (staNodes[i]);
    }

  // Set up the internet stack
  stack.Install (apNodes);
  for (uint32_t i = 0; i < apNodeCount; i++)
    {
      stack.Install (staNodes[i]);
    }

  // Assign addresses to the nodes
  Ipv4InterfaceContainer wifiApInterface[apNodeCount];
  Ipv4InterfaceContainer wifiStaInterface[apNodeCount];

  /// Create IPs for this loop (the 6th digit will be altered per
     loop)
  char base_ip[16];
  uint8_t sect_1 = 10;
  uint8_t sect_2 = 1;
  uint8_t sect_3 = 1;
  uint8_t sect_4 = 0;

  for (uint32_t i = 0; i < apNodeCount; i++)
    {
      /// Update the base_ip to use the 4 sectors
      snprintf (base_ip, 16, "%d.%d.%d.%d", sect_1, sect_2, sect_3,
         sect_4);

      /// Update the address helper's base IP
      address.SetBase (base_ip, "255.255.255.0");

      /// Assign IP addresses to the i-th group nodes
      wifiApInterface[i] = address.Assign (apNetDevices[i]); // The i
         -th contains one AP node
      wifiStaInterface[i] =
          address.Assign (staNetDevices[i]); // The i-th contains
             staNodeCount Station nodes

      /// Update sect_3 so that base IP is different next loop
      sect_3++;
    }

  // Create the UDP Echo application and all the necessary parts
  //// PS.: A lot of these cannot be created inside a FOR loop, there
```

```
        is no default constructor. Check apNodeCount.
129
130   /// Create the servers on the AP nodes
131   UdpEchoServerHelper echoServer_0 (9);
132   UdpEchoServerHelper echoServer_1 (9);
133   UdpEchoServerHelper echoServer_2 (9);
134   UdpEchoServerHelper echoServer_3 (9);
135   UdpEchoServerHelper echoServer_4 (9);
136   UdpEchoServerHelper echoServer_5 (9);
137   UdpEchoServerHelper echoServer_6 (9);
138   UdpEchoServerHelper echoServer_7 (9);
139   UdpEchoServerHelper echoServer_8 (9);
140   UdpEchoServerHelper echoServer_9 (9);
141
142   ApplicationContainer serverApps[apNodeCount];
143   serverApps[0] = echoServer_0.Install (apNodes.Get (0));
144   serverApps[1] = echoServer_1.Install (apNodes.Get (1));
145   serverApps[2] = echoServer_2.Install (apNodes.Get (2));
146   serverApps[3] = echoServer_3.Install (apNodes.Get (3));
147   serverApps[4] = echoServer_4.Install (apNodes.Get (4));
148   serverApps[5] = echoServer_5.Install (apNodes.Get (5));
149   serverApps[6] = echoServer_6.Install (apNodes.Get (6));
150   serverApps[7] = echoServer_7.Install (apNodes.Get (7));
151   serverApps[8] = echoServer_8.Install (apNodes.Get (8));
152   serverApps[9] = echoServer_9.Install (apNodes.Get (9));
153
154   for (uint32_t i = 0; i < apNodeCount; i++)
155     {
156       serverApps[i].Start (Seconds (1));
157       serverApps[i].Stop (Seconds (globalStopTime));
158     }
159
160   /// Create the clients on the Station nodes
161   UdpEchoClientHelper echoClient_0 (wifiApInterface[0].GetAddress (0)
        , 9);
162   UdpEchoClientHelper echoClient_1 (wifiApInterface[1].GetAddress (0)
        , 9);
163   UdpEchoClientHelper echoClient_2 (wifiApInterface[2].GetAddress (0)
        , 9);
164   UdpEchoClientHelper echoClient_3 (wifiApInterface[3].GetAddress (0)
        , 9);
165   UdpEchoClientHelper echoClient_4 (wifiApInterface[4].GetAddress (0)
        , 9);
166   UdpEchoClientHelper echoClient_5 (wifiApInterface[5].GetAddress (0)
        , 9);
167   UdpEchoClientHelper echoClient_6 (wifiApInterface[6].GetAddress (0)
        , 9);
168   UdpEchoClientHelper echoClient_7 (wifiApInterface[7].GetAddress (0)
        , 9);
169   UdpEchoClientHelper echoClient_8 (wifiApInterface[8].GetAddress (0)
        , 9);
```

```
170   UdpEchoClientHelper echoClient_9 (wifiApInterface[9].GetAddress (0)
          , 9);
171
172   UintegerValue maxPackets = 3;
173   TimeValue interval = Seconds (1);
174   UintegerValue packetSize = 64;
175
176   echoClient_0.SetAttribute ("MaxPackets", maxPackets);
177   echoClient_0.SetAttribute ("Interval", interval);
178   echoClient_0.SetAttribute ("PacketSize", packetSize);
179
180   echoClient_1.SetAttribute ("MaxPackets", maxPackets);
181   echoClient_1.SetAttribute ("Interval", interval);
182   echoClient_1.SetAttribute ("PacketSize", packetSize);
183
184   echoClient_2.SetAttribute ("MaxPackets", maxPackets);
185   echoClient_2.SetAttribute ("Interval", interval);
186   echoClient_2.SetAttribute ("PacketSize", packetSize);
187
188   echoClient_3.SetAttribute ("MaxPackets", maxPackets);
189   echoClient_3.SetAttribute ("Interval", interval);
190   echoClient_3.SetAttribute ("PacketSize", packetSize);
191
192   echoClient_4.SetAttribute ("MaxPackets", maxPackets);
193   echoClient_4.SetAttribute ("Interval", interval);
194   echoClient_4.SetAttribute ("PacketSize", packetSize);
195
196   echoClient_5.SetAttribute ("MaxPackets", maxPackets);
197   echoClient_5.SetAttribute ("Interval", interval);
198   echoClient_5.SetAttribute ("PacketSize", packetSize);
199
200   echoClient_6.SetAttribute ("MaxPackets", maxPackets);
201   echoClient_6.SetAttribute ("Interval", interval);
202   echoClient_6.SetAttribute ("PacketSize", packetSize);
203
204   echoClient_7.SetAttribute ("MaxPackets", maxPackets);
205   echoClient_7.SetAttribute ("Interval", interval);
206   echoClient_7.SetAttribute ("PacketSize", packetSize);
207
208   echoClient_8.SetAttribute ("MaxPackets", maxPackets);
209   echoClient_8.SetAttribute ("Interval", interval);
210   echoClient_8.SetAttribute ("PacketSize", packetSize);
211
212   echoClient_9.SetAttribute ("MaxPackets", maxPackets);
213   echoClient_9.SetAttribute ("Interval", interval);
214   echoClient_9.SetAttribute ("PacketSize", packetSize);
215
216   /// Create the client apps (these also cannot be created insider a
          FOR loop)
217   ApplicationContainer clientApps[apNodeCount];
218   double startTimeClientApp = 2;
```

```
219
220   clientApps[0] = echoClient_0.Install (staNodes[0]);
221   clientApps[1] = echoClient_1.Install (staNodes[1]);
222   clientApps[2] = echoClient_2.Install (staNodes[2]);
223   clientApps[3] = echoClient_3.Install (staNodes[3]);
224   clientApps[4] = echoClient_4.Install (staNodes[4]);
225   clientApps[5] = echoClient_5.Install (staNodes[5]);
226   clientApps[6] = echoClient_6.Install (staNodes[6]);
227   clientApps[7] = echoClient_7.Install (staNodes[7]);
228   clientApps[8] = echoClient_8.Install (staNodes[8]);
229   clientApps[9] = echoClient_9.Install (staNodes[9]);
230
231   for (uint32_t i = 0; i < apNodeCount; i++)
232     {
233       clientApps[i].Start (Seconds (startTimeClientApp));
234       clientApps[i].Stop (Seconds (globalStopTime));
235     }
236
237   // Populate the IPv4 routing tables for this network
238   Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
239
240   // Force the Simulator to stop if the given time is reached
241   Simulator::Stop (Seconds (globalStopTime));
242
243   // Final steps
244   if (tracing == true)
245     {
246       /// Wireshark tracing (using PCAP files)
247       for (uint32_t i = 0; i < apNodeCount; i++)
248         {
249           phy.EnablePcap ("experiment_v6_PCAP_APnode_" + std::
                to_string (i),
250                           apNetDevices[i].Get (0));
251         }
252
253       /// Create a XML (NetAnim) file for the simulation
254       AnimationInterface anim ("experiment_v6_NetAnim.xml");
255       anim.SetMaxPktsPerTraceFile (1000000); // One million
256
257       /// Set the base coordinates for the animation
258       double x = 20;
259       double y = x;
260
261       /// Create the tools to add the Station nodes to the animation
             randomly around the AP Node
262       double random_x = 0;
263       double random_y = 0;
264       double horizontal_lower_bound = x - 5;
265       double horizontal_upper_bound = x + 5;
266       double vertical_lower_bound = y - 10;
267       double vertical_upper_bound = y + 10;
```

```
268         std::default_random_engine dre;
269
270         /// Add the nodes to the animation
271         for (uint32_t i = 0; i < apNodeCount; i++)
272           {
273             anim.SetConstantPosition (apNodes.Get (i), x, y);
274
275             // Create the distributions here to take updated values
                    each loop
276             std::uniform_real_distribution<double> horizontal_urd (
                    horizontal_lower_bound,horizontal_upper_bound);
277             std::uniform_real_distribution<double> vertical_urd (
                    vertical_lower_bound,vertical_upper_bound);
278
279             for (uint32_t j = 0; j < staNodeCount; j++)
280               {
281                 // Randomize (x, y)
282                 random_x = horizontal_urd (dre);
283                 random_y = vertical_urd (dre);
284
285                 // Add the Station Nodes from the i-th AP node to the
                        animation
286                 anim.SetConstantPosition (staNodes[i].Get (j), random_x
                        , random_y);
287               }
288
289             // Update all the variables for the next group
290             x = x + 30;
291             y = x;
292
293             horizontal_lower_bound = x - 5;
294             horizontal_upper_bound = x + 5;
295             vertical_lower_bound = y - 10;
296             vertical_upper_bound = y + 10;
297           }
298
299         /// Set up the Flow Monitor and its helper
300         Ptr<FlowMonitor> flowMonitor;
301         FlowMonitorHelper flowHelper;
302         flowMonitor = flowHelper.InstallAll ();
303
304         /// Change the Simulator stop time to allow for some cleanup
305         /// time because of the FlowMonitor
306         /// PS.: the apps containers should remain at their normal stop
                time,
307         ///      only the Simulator has to stop later
308         Simulator::Stop (Seconds (globalStopTime + 3));
309         Simulator::Run ();
310
311         flowMonitor->SerializeToXmlFile ("flow_monitoring_v6.xml", true
                , true);
```

```
312
313       Simulator::Destroy ();
314     }
315   else
316     {
317       /// Run and then destroy the simulator
318       Simulator::Run ();
319       Simulator::Destroy ();
320     }
321
322   return 0;
323 }
```