UNIVERSIDADE FEDERAL DE SANTA CATARINA
PROGRAMA DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

René Nolio Santa Cruz

CONTROLE DE ACESSO EM AMBIENTES INTELIGENTES COM FOG E REDES NEURAIS

Florianópolis

2019

René Nolio Santa Cruz

CONTROLE DE ACESSO EM AMBIENTES INTELIGENTES COM FOG E REDES NEURAIS

Esta Monografia foi julgada aprovada para a obtenção do Título de "Bacharel em Ciências da Computação", e aprovada em sua forma final pelo Programa de Graduação em Ciência da Computação.

Florianópolis, 9 de dezembro de 2019.

Prof. José Francisco D. de G. C. Fletes
Coordenador do Curso

Prof. Dr. Carlos Becker Westphall
Orientador

Banca Examinadora:

Prof. Dr. Carlos Becker Westphall
Presidente

Me. Hugo Vaz Sampaio
Coorientador

Prof[a]. Dr[a]. Carla Merkle Westphall

Prof. Dr. Elder Rizzon Santos

RESUMO

Avanços na área de computação têm transformado a nuvem num *commodity*, onde recursos podem ser alugados com base na demanda. Redes IoT surgem visando integrar dispositivos (ou "coisas") à Internet, porém, dispositivos IoT tipicamente possuem pouco espaço de armazenamento e podem gerar grande quantidade de dados. Redes IoT foram integradas à nuvem com o objetivo de usufruir sobre as propriedades elásticas da mesma; dando origem ao paradigma de *fog*, que procura pré-processar os dados antes de enviá-los à nuvem. Por outro lado, técnicas de IA se mostraram eficientes em diversas áreas, em especial para classificação e predição de dados. Tendo em vista estes conceitos, é proposto um modelo de redes neurais na fog que, a partir dos dados de acesso em um condomínio inteligente, são estimados os horários em que as residências se encontram desocupadas. Este conhecimento pode ser utilizado para uma infinidade de otimizações e para melhorar a qualidade de vida dos moradores. A viabilidade deste modelo é demonstrada por meio de um protótipo de rede *fog*.

Palavras-chave: gerenciamento de controle de acesso, computação em névoa, Internet das coisas, inteligência artificial, redes neurais.

LISTA DE FIGURAS

Figura 1 Relação entre os paradigmas de cloud, fog e IoT. Os *end devices* podem ser dispositivos IoT. (IORGA; MARTIN; FELDMAN, 2018) 7
Figura 2 Papéis principais num ambiente de nuvem. (JENNINGS; STADLER, 2015) 11
Figura 3 Níveis de abstração na nuvem. (JENNINGS; STADLER, 2015) 11
Figura 4 Componentes principais de um neurônio artificial. (LUGER, 2009) 15
Figura 5 Neurônios artificiais de McCulloch–Pitts que computam as funções lógicas AND e OR. (LUGER, 2009) 15
Figura 6 Função lógica XOR. Não é possível separar os pontos pretos dos brancos por meio de uma reta. (LUGER, 2009) 17
Figura 7 Superfície de erro representada em duas das n dimensões. A constante c é a taxa de aprendizado. (LUGER, 2009) 17
Figura 8 *Backpropagation* em uma rede conexionista com uma camada intermediária. (LUGER, 2009) 18
Figura 9 Arquitetura proposta. (CHOUBEY et al., 2015) 21
Figura 10 Arquitetura geral do framework EiF. (AN et al., 2019) 22
Figura 11 Framework do sistema proposto. (SOOD; MAHAJAN, 2019) 23
Figura 12 Rede neural com *backpropagation* utilizada para prever crises de hipertensão. (SOOD; MAHAJAN, 2019) 23
Figura 13 Visão geral do dispositivo IoT proposto. (SAMPAIO et al., 2019) 24
Figura 14 Visão geral do modelo proposto. (CHEN; AZHARI; LEU, 2018) 25
Figura 15 Visão geral do modelo proposto (fonte própria). 26
Figura 16 Visão geral do condomínio inteligente considerado para a proposta (fonte própria). 27
Figura 17 Fluxo geral das mensagens no sistema proposto (fonte própria). 27
Figura 18 Interface do XCTU com ambos dispositivos e suas respectivas configurações (fonte própria). 29
Figura 19 Conexões entre o Arduíno e os demais componentes do dispositivo final (fonte própria). 30
Figura 20 Exemplo de pacote de requisição de transmissão (fonte própria). 31
Figura 21 Estrutura do pacote de requisição de transmissão (fonte própria). 31
Figura 22 Transmissão e processamento do pacote da figura 20 (fonte própria). 32
Figura 23 Estrutura do pacote de recebimento (fonte própria). 32
Figura 24 Interface da ferramenta XCTU, com um exemplo de pacote de transmissão (fonte própria). 33
Figura 25 Esquema do banco de dados MySQL (fonte própria). 34
Figura 26 Modelo da rede neural desenvolvida (fonte própria). 39
Figura 27 Acurácia obtida pela rede neural em relação à quantidade de épocas (fonte própria). 40
Figura 28 Perda da rede neural em relação à quantidade de épocas (fonte própria). 40

LISTA DE TABELAS

Tabela 1 Modelo de McCulloch–Pitts para AND lógico. (LUGER, 2009) 16
Tabela 2 Revisão bibliográfica indicando a quantidade de resultados por palavras chave (fonte própria). 20
Tabela 3 Cenários simulados, com os seus respectivos cronogramas e quantidade de residências (fonte própria). 35
Tabela 4 Alguns modelos de redes neurais com as suas respectivas acurácias (fonte própria). 38

LISTA DE ABREVIATURAS E SIGLAS

Edges Bordas da rede 7
Fog *Fog computing* ou computação em névoa 7
IA Inteligência artificial 7
IoT *Internet of Things* ou Internet das coisas 7
ML *Machine Learning* ou aprendizado de máquina 7
Nuvem *Cloud computing* ou computação na nuvem 7
QoS *Quality of Service* ou qualidade do serviço 7
RFID *Radio-frequency identification* ou identificação por rádio-frequência 7
SLA *Service Level Agreement* ou contrato de nível de serviço 7
PAN ID *Personal Area Network ID* ou identificador de rede de área pessoal 7
ASCII *American Standard Code for Information Interchange* ou Código Padrão Americano para Troca de Informações 7
CSV *Comma-separated values*, um padrão utilizado para armazenar dados tabulares em um arquivo de texto 7

SUMÁRIO

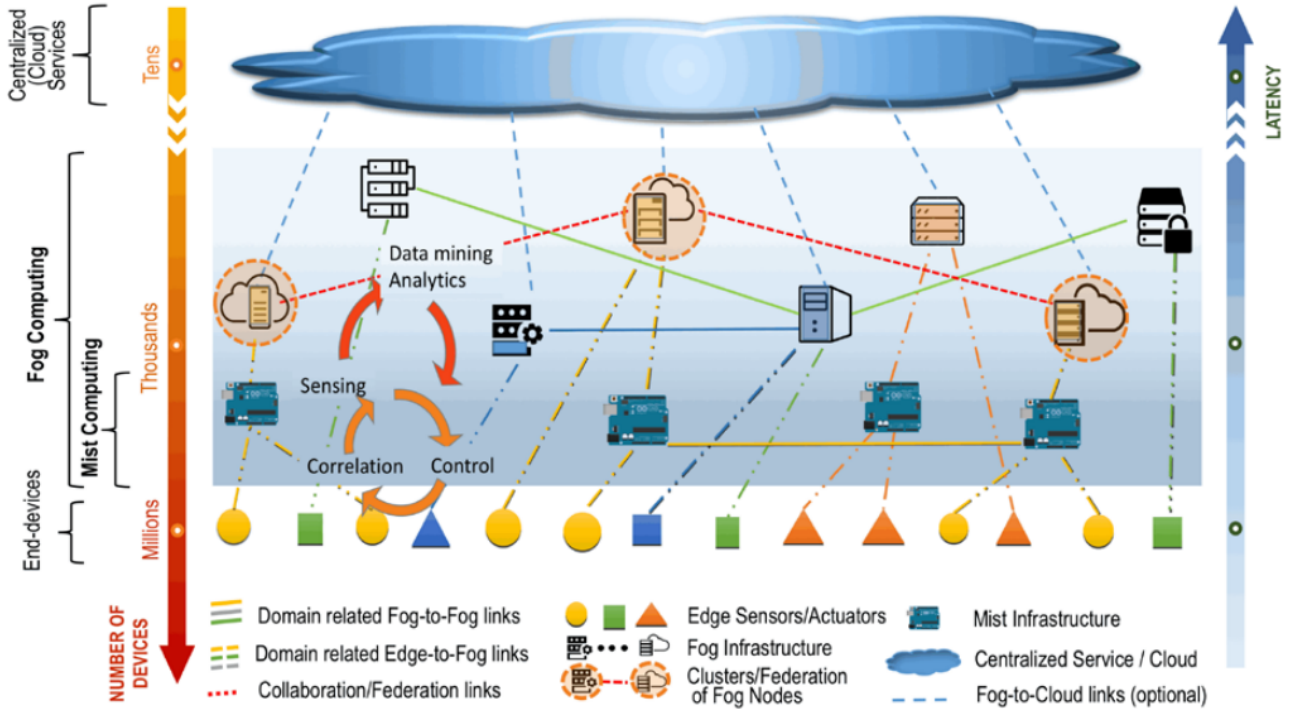
1 INTRODUÇÃO 7
1.1 OBJETIVO GERAL 8
1.2 OBJETIVOS ESPECÍFICOS 8
1.3 JUSTIFICATIVA 8
1.4 MÉTODO DE PESQUISA E TRABALHO 8
1.5 ORGANIZAÇÃO DESTE RELATÓRIO 9
2 CONCEITOS FUNDAMENTAIS 10
2.1 CLOUD COMPUTING 10
2.2 INTERNET OF THINGS 12
2.3 FOG COMPUTING 12
2.4 INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL 13
2.5 REDES NEURAIS 14
2.5.1 Keras 18
2.5.2 Tensorflow 19
3 REVISÃO BIBLIOGRÁFICA E ESTADO DA ARTE 20
3.1 POWER EFFICIENT, BANDWIDTH OPTIMIZED AND FAULT TOLERANT SENSOR MANAGEMENT FOR IOT IN SMART HOME 21
3.2 EIF: TOWARD AN ELASTIC IOT FOG FRAMEWORK FOR AI SERVICES 21
3.3 IOT-FOG BASED HEALTHCARE FRAMEWORK TO IDENTIFY AND CONTROL HYPERTENSION ATTACK 22
3.4 AUTONOMIC IOT BATTERY MANAGEMENT WITH FOG COMPUTING 24
3.5 DESIGN AND IMPLEMENTATION OF A POWER CONSUMPTION MANAGEMENT SYSTEM FOR SMART HOME OVER FOG-CLOUD COMPUTING 25
4 DESENVOLVIMENTO DA PROPOSTA 26
4.1 DESENVOLVIMENTO DO PROTÓTIPO DE NÓ IOT 28
4.1.1 Desenvolvimento do nó final 28
4.1.2 Protocolo *Zigbee* 28
4.2 DESENVOLVIMENTO DO SISTEMA FOG 29
4.2.1 Montagem de pacotes 30
5 GERENCIAMENTO AUTONÔMICO COM IA 34
5.1 ARMAZENAMENTO DE DADOS 34
5.2 SIMULAÇÃO DE DADOS 35
5.2.1 Tratamento de dados 36
5.3 REDE NEURAL 36
5.3.1 Desenvolvimento da rede neural 36
6 CONCLUSÕES E TRABALHOS FUTUROS 41
REFERÊNCIAS 42

1 INTRODUÇÃO

O paradigma de IoT procura integrar coisas à Internet, visando criar um mundo de dispositivos conectados (GUPTA et al., 2017). De acordo com Ray (2018) e Gupta et al. (2017), as coisas, ou *things*, são dispositivos que possuem interfaces de comunicação com e sem fio, são munidos de sensores, e tipicamente possuem baixo poder computacional e baixa capacidade de armazenamento; porém podem produzir quantidades imensas de dados. Exemplos de coisas incluem sensores inteligentes, dispositivos médicos, geladeiras, veículos, câmeras inteligentes e outros (GUPTA et al., 2017).

A computação na nuvem é um paradigma que consiste em compartilhar recursos computacionais por meio da Internet, geralmente com base na demanda, como um serviço ou *commodity* (JENNINGS; STADLER, 2015). De acordo com Iorga, Martin e Feldman (2018), foram propostas diversas arquiteturas que integram os paradigmas de nuvem e IoT. Uma destas abordagens, ilustrada na figura 1, consiste em transmitir os dados gerados pelos dispositivos IoT à nuvem, para o seu posterior processamento e armazenamento. Tendo em vista que, de acordo com Iorga, Martin e Feldman (2018), a quantidade de dispositivos inteligentes interconectados deverá passar de 50 bilhões até o ano 2020, esta abordagem se torna inviável, especialmente para aplicativos de tempo real.

Figura 1 – Relação entre os paradigmas de cloud, fog e IoT. Os *end devices* podem ser dispositivos IoT. (IORGA; MARTIN; FELDMAN, 2018)



Dentre os diversos desafios em aberto na área de IoT, podemos destacar a grande quantidade de dados heterogêneos gerados que, ao serem transmitidos para a nuvem, geram grande consumo de banda, alta latência e baixa qualidade de serviço (CHOUBEY et al., 2015). Em particular, a nuvem é eficiente para processar dados em *batch*, mas não é ideal para processamento em tempo real (JENNINGS; STADLER, 2015). Para amenizar estas desvantagens, foi proposto o paradigma de *fog computing*, que consiste em utilizar dispositivos de uma camada intermediária entre IoT e nuvem para pré-processar os dados.

Por outro lado, técnicas de inteligência artificial também podem ser utilizadas para amenizar os mesmos problemas. Choubey et al. (2015) utilizam redes neurais para encontrar relações

entre os valores medidos por sensores IoT, com o objetivo de reduzir a redundância de sensores e minimizar o consumo energético. Sood e Mahajan (2019) propõem um *framework* que, por meio de redes neurais, estima o risco de ataque de hipertensão a partir de dados capturados por meio de sensores IoT, permitindo notificar serviços de emergência a tempo. An et al. (2019) propõem um framework que utilizam algoritmos adaptativos distribuídos de IA para gerenciar nodos fog e coordená-los com a nuvem, com o objetivo de melhorar o QoS.

Desta forma, procurou-se combinar o paradigma de *fog computing* e técnicas de IA para encontrar os horários em que as residências de um condomínio estão desocupadas.

1.1 OBJETIVO GERAL

O objetivo principal é desenvolver um dispositivo IoT para gerenciar o controle de acesso de apartamentos inteligentes, assim como utilizar técnicas de inteligência artificial para encontrar os horários em que a residência está desocupada. Estes conhecimentos permitiriam aplicar várias otimizações que aumentam a qualidade de vida dos moradores e reduz desperdícios. É importante destacar que este trabalho não tem como objetivo conduzir uma análise detalhada na área de inteligência artificial, mas sim aplicar as técnicas e conhecimentos existentes em uma rede fog-IoT.

1.2 OBJETIVOS ESPECÍFICOS

Os objetivos específicos deste trabalho são:

- Propor e desenvolver um dispositivo IoT com um sensor RFID e um ambiente fog para processar os valores medidos.
- Utilizar técnicas de IA para encontrar os horários em que a residência está desocupada conforme o perfil de diferentes residências.
- Validar este projeto por meio de testes, com um protótipo de rede fog, assim como avaliar a análise dos dados, obtidos por meio de simulações, gerada pela rede neural.

1.3 JUSTIFICATIVA

Dentre os principais problemas encontrados em redes fog podemos destacar o grande volume de dados gerados pelos dispositivos IoT, que devem ser transmitidos pela rede. Por outro lado, técnicas de IA demonstraram ser eficientes em diversas áreas, e em especial, para lidar com problemas de classificação e predição de dados.

Ao aplicar técnicas de IA na fog, é possível inferir horários nos quais as residências estão desocupadas, o que permitiria o gerenciamento autonômico de sistemas IoT da residência, por exemplo, otimizar o consumo energético dos dispositivos IoT ao otimizar o tempo de *sleep* dos mesmos.

1.4 MÉTODO DE PESQUISA E TRABALHO

Para o desenvolvimento deste trabalho de conclusão de curso, os principais métodos de pesquisa e trabalho são:

1. Pesquisa de livros e artigos recentes (preferencialmente a partir de 2015) nas áreas de IoT,

IA, fog e demais áreas relevantes, no contexto de ambientes inteligentes (casas, estufas, ambientes inteligentes).

2. Análise detalhada dos artigos mais correlatos e focados nesta proposta, de modo a compreender corretamente os conceitos sobre diversos pontos de vista.

3. Validação da solução proposta através de um protótipo, utilizando dados obtidos por meio de simulações.

4. Implementação do protótipo proposto, realização de testes e análise dos resultados obtidos.

1.5 ORGANIZAÇÃO DESTE RELATÓRIO

Este relatório está organizado da seguinte maneira: No capítulo 1 é apresentada a introdução; as seções 1.1 e 1.2 detalham os objetivos gerais e específicos deste trabalho, respectivamente; a seção 1.3 apresenta a justificativa para o desenvolvimento deste trabalho e a seção 1.4 descreve o método de pesquisa e trabalho seguido.

O capítulo 2 apresenta os conceitos fundamentais utilizados como base teórica, como cloud, fog, IoT, inteligência artificial e redes neurais. No capítulo 3, foram apresentados os resultados da revisão bibliográfica e 5 artigos correlatos foram detalhados.

O capítulo 4 apresenta a proposta. A seção 4.1 descreve como o protótipo de nó IoT foi desenvolvido, o protocolo utilizado e o sistema fog. O capítulo 5 descreve o desenvolvimento do sistema de gerenciamento com IA, como os dados foram armazenados, como foram simulados, e o desenvolvimento da rede neural utilizada.

O capítulo 6 cita as conclusões e trabalhos futuros que podem ser desenvolvidos com base neste trabalho. Finalmente, são descritas as referências bibliográficas.

2 CONCEITOS FUNDAMENTAIS

Neste capítulo são apresentados brevemente alguns conceitos fundamentais necessários à confecção deste trabalho e as principais técnicas e tecnologias já conhecidas e utilizadas nos meios acadêmico e comercial, nas áreas de *cloud*, fog e IoT, além de uma breve introdução à inteligência artificial e à área de redes neurais.

2.1 CLOUD COMPUTING

Na última década, avanços na computação como um serviço e nas tecnologias de virtualização, permitiram a construção de *data centers* lucrativos em alta escala, que executam grande parte dos aplicativos na Internet e processamento em *backend*, permitindo que a infraestrutura de data centers seja alugada a terceiros. Segundo Jennings e Stadler (2015), emerge, então, o paradigma de Computação em Nuvem (*Cloud Computing*), no qual um conjunto de recursos computacionais é compartilhado entre aplicativos que o acessam pela Internet. Este conceito pode também se referir a hardware e software de sistema que reside nos data centers que hospedam tais aplicativos. Os objetivos dos provedores de nuvem estão centrados no uso eficiente dos recursos, dentro de limites estabelecidos por um Acordo de Nível de Serviço, ou *Service Level Agreement* (SLA), que é um contrato formal entre o provedor e o usuário, cujo objetivo é definir os aspectos funcionais e não funcionais do serviço em termos quantitativos (JENNINGS; STADLER, 2015).

De acordo com a sua infraestrutura e o modelo de provisionamento de recursos, podemos classificar a nuvem em quatro principais categorias (BUYYA et al., 2009):

- Nuvem pública: a nuvem abrange o provisionamento de recursos a terceiros alugando-os com base no uso.
- Nuvem privada: compreende as infraestruturas privadas mantidas e utilizadas por indivíduos e organizações.
- Nuvem híbrida: pode ser definida como o cenário em que uma organização estende a sua nuvem privada ao alugar parte dos seus recursos de uma nuvem pública.
- Nuvens comunitárias: nesta categoria, os recursos são contribuídos por vários indivíduos e/ou organizações de forma descentralizada.

De acordo com Jennings e Stadler (2015), de forma geral, podemos destacar três papéis principais num ambiente de nuvem, conforme demonstrado na figura 2:

- Provedor da nuvem: Gerencia os *data centers*, provendo estes recursos de forma abstrata para os usuários da nuvem ou usuários finais e é responsável por cumprir os SLAs.
- Usuário da nuvem: utiliza serviços da nuvem para oferecer aplicativos para os usuários finais, procurando minimizar custos, maximizar lucros e manter em sintonia a quantidade de recursos requerida pelos seus clientes com a quantidade alugada do provedor da nuvem.
- Usuário final: Num ambiente comercial, é o cliente final. Apesar de geralmente não atuar diretamente no gerenciamento da nuvem, pode influenciar as decisões tomadas pelo usuário da nuvem e provedor da nuvem.

De acordo com Dinh et al. (2013), dependendo do nível de abstração no qual o serviço é oferecido, ambientes de nuvem pública podem ser classificados em três principais categorias, conforme demonstrado na figura 3:

Figura 2 – Papéis principais num ambiente de nuvem. (JENNINGS; STADLER, 2015)

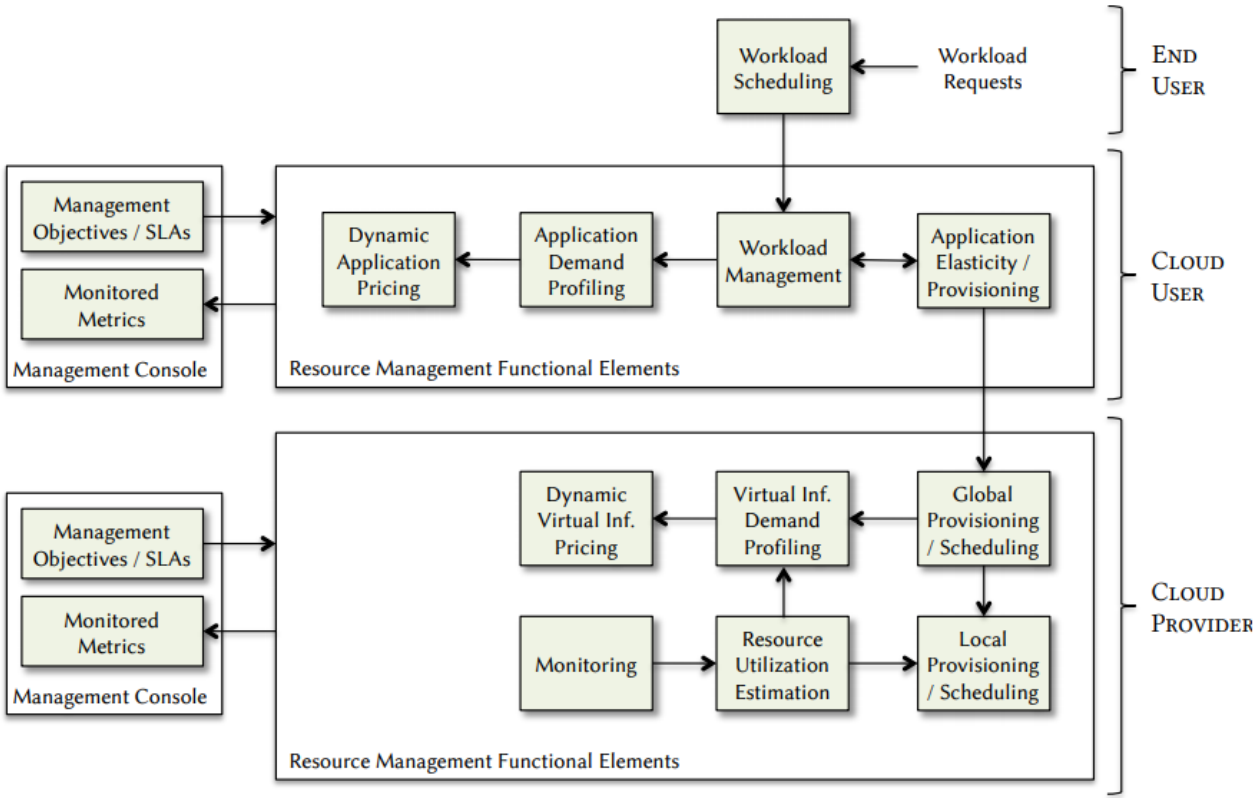
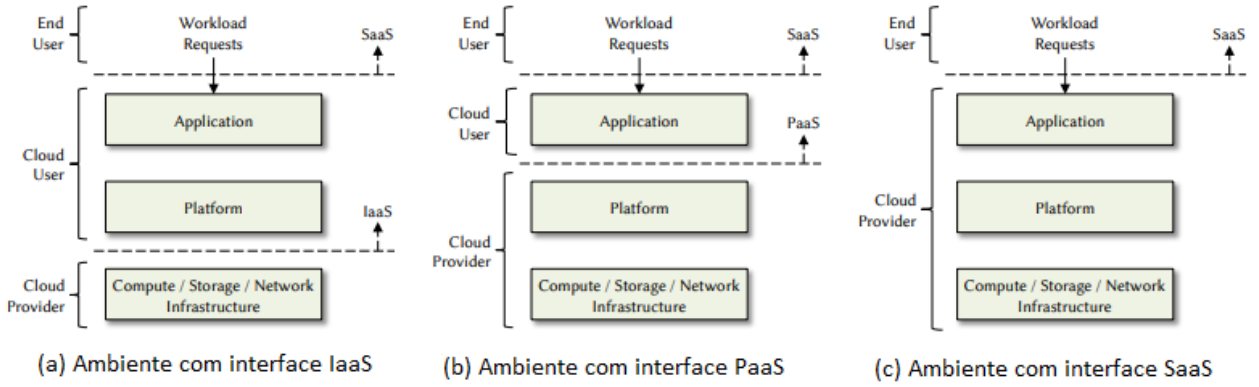


Figura 3 – Níveis de abstração na nuvem. (JENNINGS; STADLER, 2015)



- IaaS (*Infrastructure-as-a-Service*, ou infraestrutura como serviço): O provedor da nuvem irá prover recursos procurando cumprir com o SLA acordado com o usuário da nuvem. O SLA define formal e quantitativamente aspectos dos serviços oferecidos, como disponibilidade, desempenho, tolerância a falhas, entre outros. O provedor da nuvem pode oferecer diferentes níveis de serviço e priorizar certos aspectos dos serviços dependendo do SLA e condições operacionais. O usuário da nuvem geralmente possui um SLA com os usuários finais e procura explorar a elasticidade de recursos presente na nuvem para acomodar as necessidades dos seus clientes. O custo para o cliente geralmente é baseado na quantidade de recursos utilizados; a infraestrutura pode ser expandida ou encolhida dinamicamente conforme necessário.

- PaaS (*Platform-as-a-Service*, ou plataforma como serviço): Oferece um ambiente integrado para desenvolver, testar e implantar aplicativos. Exemplos deste modelo de serviço incluem *Google App Engine* e *Microsoft Azure*.

- SaaS (*Software-as-a-Service*, ou software como serviço): É capaz de distribuir software com requisitos específicos. Os usuários podem acessar aplicativos remotamente e o custo é baseado na quantidade de recursos utilizados. Exemplos deste modelo de serviço incluem *Salesforce* e *Microsoft's Live Mesh.*

2.2 INTERNET OF THINGS

O paradigma de IoT tem como objetivo integrar à Internet "coisas" (ou *"things"*, em inglês), como dispositivos médicos, geladeiras, câmeras e sensores. Este paradigma permite novas formas de interação entre coisas e pessoas, e permite melhorar a qualidade de vida e a utilização de recursos (GUPTA et al., 2017). Dá origem a uma visão de um mundo de dispositivos e pessoas conectados. De acordo com Gupta et al. (2017), estima-se que até 2025 IoT terá um impacto econômico de 11 trilhões de dólares, e cerca de um trilhão de dispositivos IoT serão implantados. Os desafios mais importantes são heterogeneidade, escalabilidade, interoperabilidade, segurança e privacidade (YASUMOTO; YAMAGUCHI; SHIGENO, 2016). Para refletir situações reais, processamento em tempo real de dados é requerido; porém, abordagens baseadas em nuvem possuem atrasos consideráveis, diminuindo a qualidade do serviço e desperdiçam recursos da nuvem e banda; e com milhões de dispositivos, esta abordagem é inviável (YASUMOTO; YAMAGUCHI; SHIGENO, 2016),(GUPTA et al., 2017).

As limitações de processamento e armazenamento dos dispositivos de IoT, combinadas com o alto processamento de dispositivos *edge*, deram origem à fog, ou computação em névoa, que estende serviços da nuvem à edge, resultando em redução de latência. Dados passam por vários dispositivos entre o seu ponto de origem e o destino, e é importante utilizar a capacidade computacional e de armazenamento destes dispositivos intermediários (ex: gateways, roteadores, etc) (GUPTA et al., 2017).

2.3 FOG COMPUTING

Fog computing ou computação em névoa (doravante referenciado apenas como fog) pode ser definido como um paradigma computacional distribuído, que estende os serviços oferecidos pela nuvem às *edges* (ou bordas) da rede (GUPTA et al., 2017); Facilita o gerenciamento de serviços de rede, computação e armazenamento entre *data centers* e dispositivos, e foi proposto para diminuir o espaço existente entre data centers e dispositivos IoT (ALRAWAIS et al., 2017).

Fog envolve a execução de aplicativos tanto na nuvem como em dispositivos intermediários entre a nuvem e as coisas, como gateways e roteadores, por exemplo (GUPTA et al., 2017). Fog suporta mobilidade, distribuição geográfica, escalabilidade, heterogeneidade de recursos e interfaces, interação com a nuvem, análise distribuída de dados e baixa latência. Os objetivos principais deste paradigma são reduzir o volume de dados, reduzir o tráfego entre dispositivos IoT e a nuvem, diminuir a latência e melhorar a qualidade do serviço (*Quality of Service* ou QoS) (ALRAWAIS et al., 2017).

Fog procura se beneficiar tanto da proximidade dos dispositivos *edge* aos dispositivos *endpoint* (ou pontos de extremidade) quanto da escalabilidade de recursos sob demanda da nuvem. De acordo com Gupta et al. (2017), dentre os principais benefícios oferecidos por este paradigma podemos destacar:

- Fog permite filtrar e analisar os dados gerados pelos sensores utilizando dispositivos *edge*, reduzindo drasticamente a quantidade de dados enviados à nuvem;

- Caso todos os dados gerados pelos sensores forem enviados diretamente à nuvem, pode surgir um gargalo de desempenho. Como fog permite filtrar e processar uma quantidade

significativa de dados dos sensores, a arquitetura de processamento de dados pode ser modelada de forma distribuída e escalável; e,

- O processo de comunicação entre os sensores e a nuvem pode ser lento ou estar indisponível caso ocorram problemas de comunicação. Ao processar os dados localmente na fog podemos obter uma redução considerável na latência, permitindo o processamento de dados em tempo real.

Fog pode ser também ser considerada como uma extensão da nuvem que, ao mesmo tempo, introduz novos desafios de segurança e privacidade e oferece uma plataforma ideal para tratar múltiplas questões de segurança e privacidade presentes em redes IoT. O poder computacional de nós fog pode ser utilizado para prover criptografia para os dispositivos IoT localizados abaixo desse nó na topologia de rede e, desta forma, oferecer um novo nível de segurança às redes IoT. (ALRAWAIS et al., 2017).

2.4 INTRODUÇÃO À INTELIGÊNCIA ARTIFICIAL

Na última década, sistemas de inteligência artificial e aprendizado de máquina obtiveram desempenho excelente em tarefas anteriormente consideradas computacionalmente impossíveis. Foram implementados em áreas como reconhecimento de voz, classificação de imagens e jogos de tabuleiro; mas também em áreas críticas, como finanças, medicina, carros autônomos e recomendações de conteúdo. Estes sistemas devem satisfazer certos critérios, como imparcialidade, confiabilidade, segurança, privacidade e usabilidade (DOSILOVIC; BRCIC; HLUPIC, 2018).

Dosilovic, Brcic e Hlupic (2018) classificam os diversos modelos de inteligência artificial de acordo com a opacidade em três categorias:

- Modelos de IA de baixa complexidade, como modelos lineares, árvores de decisão e regras; estes modelos são comuns no meio acadêmico e comercial.

- Modelos opacos cuja complexidade não permite desvendar facilmente a lógica nas predições; logo estes modelos são considerados caixas pretas. Esta categoria inclui redes neurais artificiais, *boosted trees* e florestas aleatórias, entre outros.

- Modelos híbridos, que combinam os dois modelos anteriores procurando balancear interpretabilidade e previsibilidade.

Como o aprendizado engloba capacidades de inteligência humana, para criar uma IA devemos lidar com problemas de linguagem natural, raciocínio automatizado e aprendizado de máquina. Segundo Luger (2009), podemos classificar os diversos métodos de aprendizado em quatro principais grupos:

- Baseados em símbolos, onde entidades e relacionamentos de um problema são representados por um conjunto de símbolos. Algoritmos procuram inferir generalizações lógicas e válidas que podem ser expressas por meio destes símbolos.

- Abordagens conexionistas, inspiradas pela arquitetura do cérebro humano. Representam conhecimento como padrões de atividade em redes de pequenas unidades de processamento que, ao modificarem a sua estrutura e pesos, aprendem com base em dados de treinamento. Em lugar de utilizar as generalizações oferecidas por uma linguagem simbólica, modelos conexionistas reconhecem padrões nos dados e os refletem na sua própria estrutura.

- Algoritmos genéticos, inspirados pela genética e teoria da evolução de Darwin. Começam com uma população de possíveis soluções que são avaliadas de acordo com a sua habilidade de resolver o problema, logo as melhores soluções têm maior probabilidade de sobreviver e de combinar-se entre si para gerar a próxima geração de soluções possíveis.

- Abordagens estocásticas, que são baseadas na estatística ou, mais especificamente, na regra de Bayes, onde a experiência condiciona as expectativas para interpretar novos dados. Utilizam-se principalmente modelos Markovianos para demonstrar a interpretação de conjuntos em raciocínio que afetam como a resposta de um agente pode ser encorajada ou desencorajada por feedback.

Para este trabalho, a técnica de inteligência artificial escolhida foi a abordagem conexionista. Esta técnica foi selecionada porque, de acordo com Luger (2009), redes neurais são apropriadas para reconhecimento de padrões e não requerem o uso explícito de símbolos por parte do programador, mas aprendem por meio de exemplos. Esta última característica é de grande interesse, pois seria inviável reprogramar o modelo proposto continuamente, de forma manual. Outras técnicas, como as simbólicas e estocásticas, não são de interesse para este trabalho, pois não se conhece, a priori, algum modelo matemático que represente uma solução para este problema, fazendo-se necessário um estudo detalhado sobre o sistema a fim de formular tal modelo.

2.5 REDES NEURAIS

Em particular, nas redes neurais, a escolha de um padrão de codificação pode ser crucial para o aprendizado, já que tanto os padrões quanto as conexões entre componentes são representados por valores numéricos. Uma das principais vantagens desta abordagem é que redes neurais são treinadas (ou condicionadas) em lugar de serem programadas explicitamente, podendo assim capturar detalhes do mundo real sem terem sido explicitamente programadas para reconhecê-los (LUGER, 2009).

Esta abordagem é eficiente para aplicações difíceis de representar por modelos simbólicos e permite representar problemas cujo domínio requer habilidades de percepção humana ou não possui uma sintaxe clara. Também possui degradação gradual, ou seja, a perda de elementos ou sinapses não causa falhas na rede como um todo. De acordo com Luger (2009), as principais tarefas para as quais redes neurais são mais adequadas são:

- classificação de dados;

- reconhecimento de padrões;

- recuperação de memórias, inclusive em memória associativa;

- previsão de causas a partir de efeitos;

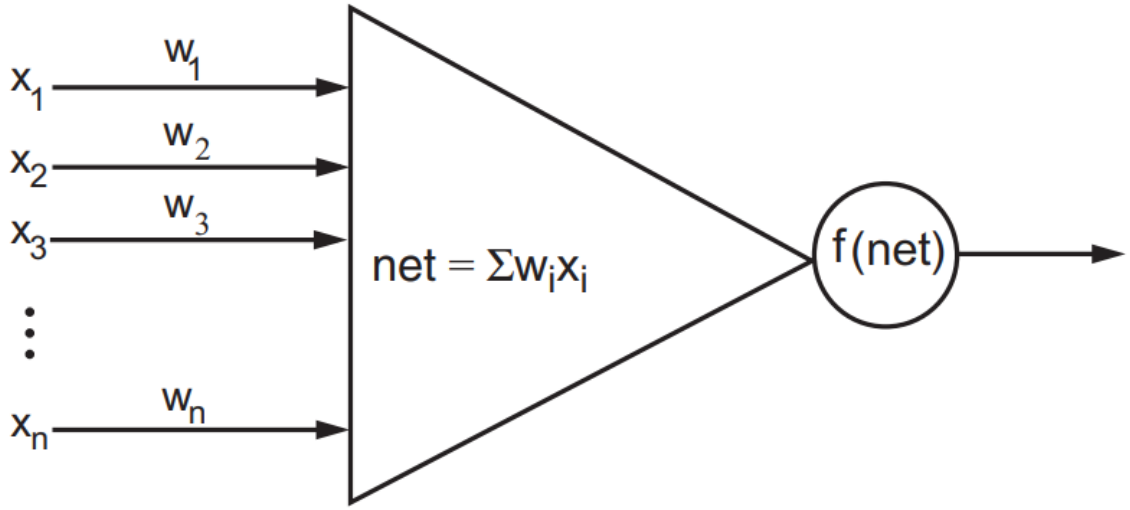
- otimizações, inclusive respeitando restrições; e,

- filtragem de ruído.

A base das redes neurais é o neurônio artificial que, de acordo com Luger (2009), possui quatro principais componentes, conforme demonstrado na figura 4:

- sinais de entrada x_i, que podem vir do meio externo ou de outros neurônios. Geralmente possuem valores no conjunto {-1,1} ou são números reais. Alguns destes sinais podem ser constantes (chamados de *bias* ou viés).

Figura 4 – Componentes principais de um neurônio artificial. (LUGER, 2009)



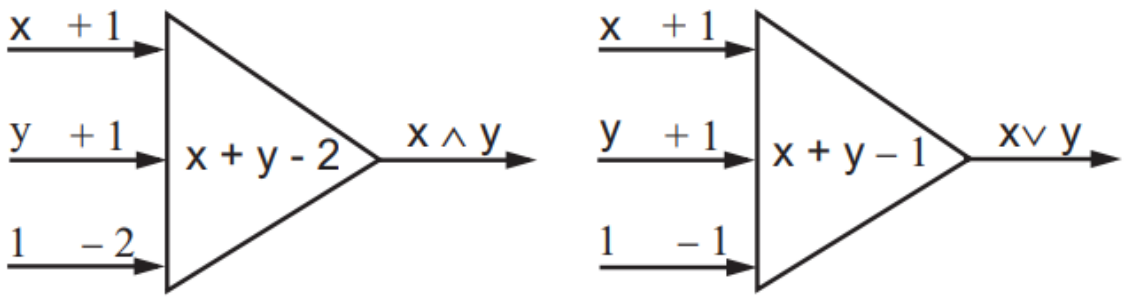
- pesos w_i, valores reais que refletem a força de uma conexão.
- nível de ativação $\Sigma w_i\ x_i$, é a soma ponderada das entradas.
- função de ativação f, que pode ser qualquer função contínua e derivável; as mais utilizadas são funções degrau, lineares, sigmodais, logarítmicas ou tangente hiperbólica. Avalia a saída do neurônio comparando o nível de ativação com algum valor limite, e o valor da saída pertence ao conjunto {-1,1} ou aos números reais.

As redes neurais como um todo possuem três características principais:

- A topologia de rede, que representa o padrão de conexões existentes entre os neurônios.
- O algoritmo de aprendizado usado.
- O esquema de codificação e decodificação de valores, utilizados na entrada e saída da rede, respectivamente.

A figura 5 e a tabela 1 demonstram uma rede perceptron, isto é, uma rede com uma única camada de neurônios em paralelo, que implementa as funções lógicas AND e OR. Este modelo foi proposto por McCulloch e Pitts em 1943.

Figura 5 – Neurônios artificiais de McCulloch–Pitts que computam as funções lógicas AND e OR. (LUGER, 2009)



x	y	x+y-2	Output
1	1	0	1
1	0	-1	-1
0	1	-1	-1
0	0	-2	-1

Tabela 1 – Modelo de McCulloch–Pitts para AND lógico. (LUGER, 2009)

O aprendizado em uma rede perceptron, pode ser supervisionado: a saída da rede neural é comparada com a saída esperada, que é fornecida por um professor (LUGER, 2009). Logo, é calculado o erro entre elas e o neurônio atualiza os pesos para tentar reduzir o erro, segundo a seguinte regra:

$$\Delta w_i = c \left(d - \text{saída} \left(\sum x_i \, w_i \right) \right) x_i$$

Onde:

- c é uma constante que determina a taxa de aprendizado.
- d é o resultado esperado, fornecido pelo professor.
- Δw_i é o ajuste de peso para o i-ésimo componente do vetor de entrada.
- saída $(\sum x_i \, w_i)$ é o valor de saída do perceptron, que é 1 ou -1.

A diferença entre a saída gerada e a esperada será 0, 2 ou -2. Para minimizar o erro médio do conjunto de treinamento, verificamos para cada componente do vetor de entrada:

- se a diferença é nula, não é necessário ajustar os pesos.
- se a saída gerada é -1 e a esperada é 1, incrementamos os pesos na i-ésima linha por $2cx_i$.
- se a saída gerada é 1 e a esperada é -1, decrementamos os pesos na i-ésima linha por $2cx_i$.

Modelos perceptron somente são capazes de resolver problemas linearmente separáveis. Um dos exemplos mais simples de não linearidade é o ou-exclusivo (XOR), representado na figura 6.

Regra do delta generalizado ou do gradiente descendente: conforme ilustrado na figura 7, é baseada em uma superfície que representa o erro cumulativo. O eixo y representa o erro acumulado e o eixo x cada possível conjunto de pesos dos neurônios. Para encontrar a direção na qual o erro é reduzido, utilizamos o vetor gradiente, procurando reduzir o erro. Uma desvantagem deste método é que pode assumir erroneamente que um ponto de mínimo local é ótimo, mesmo não sendo o mínimo global.

Se o valor da taxa de aprendizado for alto, a rede se aproxima mais rapidamente dos pontos ótimos, mas pode não convergir ou oscilar perto dos pontos de mínimo; estes problemas podem ser resolvidos usando um valor menor, porém o aprendizado será mais lento. O valor ótimo varia de acordo com a aplicação, e inclusive pode ser aprimorado usando um fator de momento ou inércia, que pode acelerar a convergência e "pular" alguns mínimos locais.

Modelos perceptron são incapazes de resolver problemas não linearmente separáveis, porém esta limitação pode ser contornada ao empilhar duas ou mais camadas de neurônios (LUGER, 2009). Redes multicamadas são conectadas em camadas, onde os neurônios de uma camada somente passam os seus valores de saída para a próxima camada. Uma rede multicamada possui três tipos de camadas, conforme demonstrado na figura 8:

Figura 6 – Função lógica XOR. Não é possível separar os pontos pretos dos brancos por meio de uma reta. (LUGER, 2009)

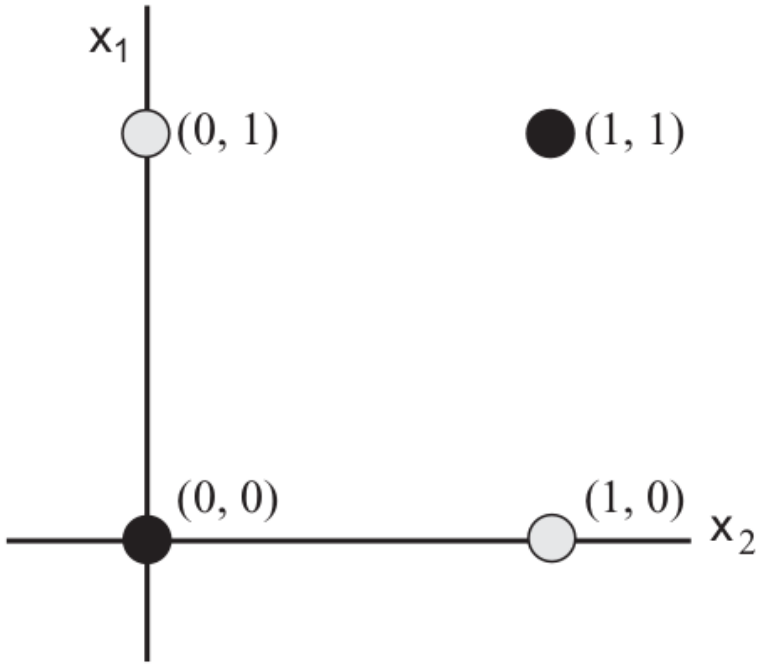
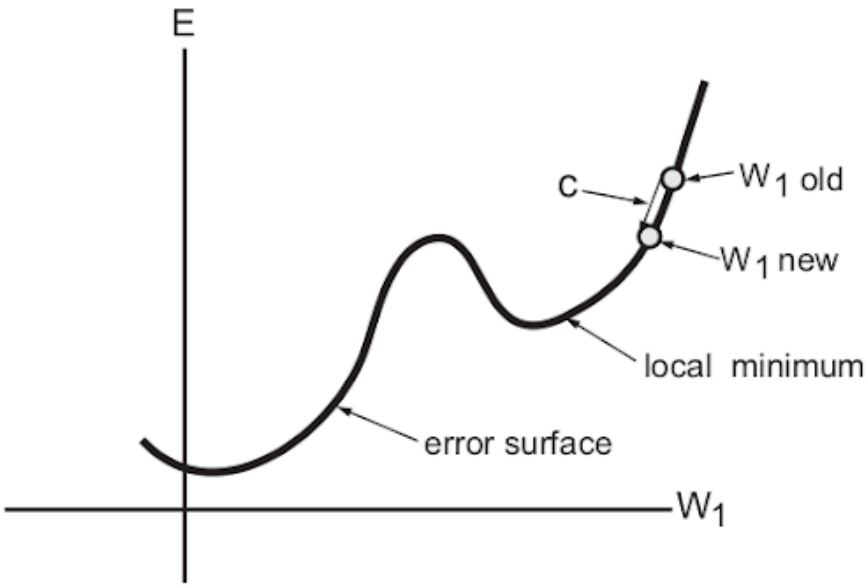


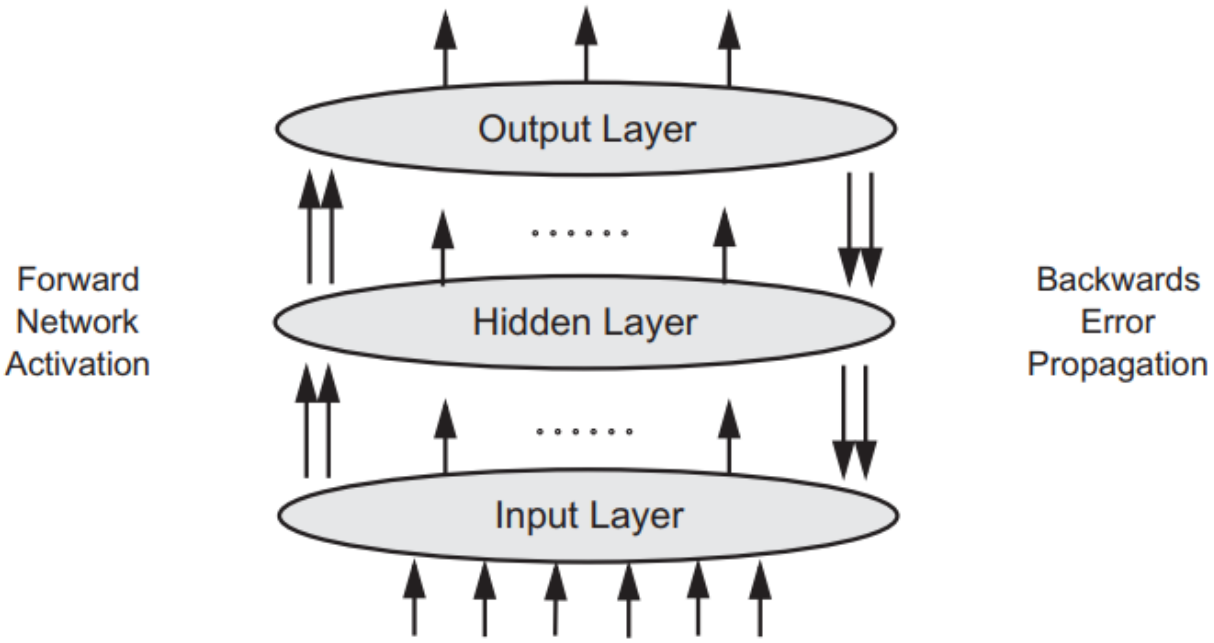
Figura 7 – Superfície de erro representada em duas das n dimensões. A constante c é a taxa de aprendizado. (LUGER, 2009)



- camada de entrada
- camadas intermediárias ou ocultas, constituídas por uma ou múltiplas camadas. O segundo caso é referido como redes neurais profundas ou *deep neural networks* (DNN), em inglês.
- camada de saída

Para ajustar os pesos dos neurônios das camadas intermediárias, o algoritmo de *backpropagation* pode ser usado. Este algoritmo é baseado na regra do delta generalizado e propaga

Figura 8 – *Backpropagation* em uma rede conexionista com uma camada intermediária. (LUGER, 2009)



o erro da camada de saída para as camadas intermediárias, avaliando a contribuição feita por cada camada para o erro total na saída. Para múltiplas camadas intermediárias este procedimento pode ser repetido recursivamente. Quando a rede converge lentamente este método pode requerer grande quantidade de poder computacional.

Para realizar o treinamento da rede, o professor separa o conjunto de dados em dois subconjuntos: 70 a 80% dos dados são usados para treinar a rede e os demais para testes e validação.

2.5.1 Keras

De acordo com Chollet et al. (2015), Keras é uma API para a programação de redes neurais em alto nível, que abstrai três linguagens de programação de redes neurais de baixo nível: TensorFlow, CNTK e Theano; sendo TensorFlow a opção selecionada por padrão. É *open-source* e escrita em python.

Oferece suporte para redes neurais rasas, convolucionais e recorrentes; é modular, extensível, flexível e suporta múltiplos *backends*. Permite descrever os modelos por meio da linguagem python, e salvá-los em um arquivo HDF5.

De acordo com Chollet et al. (2015), esta ferramenta é utilizada por grandes empresas, como Netflix e Uber, assim como *startups*. Pode ser implantada em diversas plataformas, como iOS, android, *Java Virtual Machine* e Google Cloud. Possui suporte para paralelismo em GPUs de duas formas: paralelismo de dados e de dispositivos.

Keras oferece diversas opções para tratamento de dados, permitindo normalizar os dados de cada *batch*, e por padrão os dados de treinamento são embaralhados a cada época. Também oferece otimizadores, como o gradiente descendente estocástico, que suportam momento e taxa de aprendizagem com decaimento.

2.5.2 Tensorflow

De acordo com Abadi et al. (2015), Tensorflow é uma interface, desenvolvida pela Google, para descrever algoritmos de aprendizado de máquina. Esta ferramenta já é usada em pesquisa e em diversas aplicações de aprendizado de máquina, como reconhecimento de voz, visão computacional, processamento de linguagem e classificação de objetos.

Esta ferramenta é flexível e suporta uma variedade de algoritmos de aprendizado de máquina, como algoritmos de treinamento e de inferência para redes neurais. Um programa em Tensorflow pode ser executado em diversos sistemas e dispositivos, com poucas ou nenhuma alteração. Esta característica é valiosa pois o uso de vários sistemas em larga e pequena escala necessitam de um esforço enorme de manutenção.

Tensorflow foi desenvolvido para permitir implementar e testar rapidamente novas ideias e ao mesmo tempo manter alto desempenho. Suporta paralelismo, execução distribuída, operações em GPU e uma grande gama de plataformas heterogêneas de hardware; assim como várias otimizações que aumentam o desempenho e tolerância a faltas.

Tensorflow permite representar redes neurais por meio de um grafo direcionado, descrito em C++ ou Python. Também permite adicionar extensões que oferecem algoritmos de otimização comumente utilizados, como gradiente descendente estocástico e execução parcial de uma rede neural.

3 REVISÃO BIBLIOGRÁFICA E ESTADO DA ARTE

A fim de avaliar a relevância deste trabalho, uma revisão bibliográfica foi realizada por meio das ferramentas de pesquisa *Google Scholar*, *IEEE Explorer* e *ScienceDirect*, os resultados foram obtidos ao consultar as palavras chave em inglês e considerando apenas resultados desde o ano 2015. A tabela 2 indica a quantidade de resultados obtidos nesta pesquisa em cada uma das ferramentas mencionadas anteriormente, e a coluna categoria indica a quantidade de palavras chave que foram combinadas.

Legenda para a tabela:

- AC = *Access Control*
- Fog = *Fog Computing*
- IoT = *Internet of Things*
- AI = *Artificial Intelligence*
- NN = *Neural Network*

Categoria	Palavras-chave	Google Scholar	IEEE Explorer	ScienceDirect
1	AC	136,000	64,497	7,421
1	Fog	13,500	2,107	757
1	IoT	128,000	27,127	11,954
1	AI	207,000	66,761	26,611
1	NN	239,000	52,912	70,012
2	AC, fog	6,910	3	396
2	AC, IoT	20,600	812	1,693
2	AC, AI	17,200	0	630
2	AC, NN	13,200	0	748
2	Fog, IoT	10,500	986	587
2	Fog, AI	2,520	145	163
2	Fog, NN	1,490	40	164
2	IoT, AI	22,900	2,240	2,107
2	IoT, NN	17,900	768	1,742
3	AC, fog, IoT	3,360	31	216
3	AC, fog, AI	934	3	66
3	AC, fog, NN	478	0	52
3	AC, IoT, AI	6,720	60	299
3	AC, IoT, NN	3,300	11	244
3	AC, AI, NN	4,770	129	212
4	AC, fog, IoT, AI	866	0	61
4	AC, fog, IoT, NN	444	0	48
4	AC, fog, AI, NN	281	0	29
4	AC, IoT, AI, NN	1,570	9	106
4	Fog, IoT, AI, NN	607	7	60
5	AC, fog, IoT, AI, NN	264	0	27

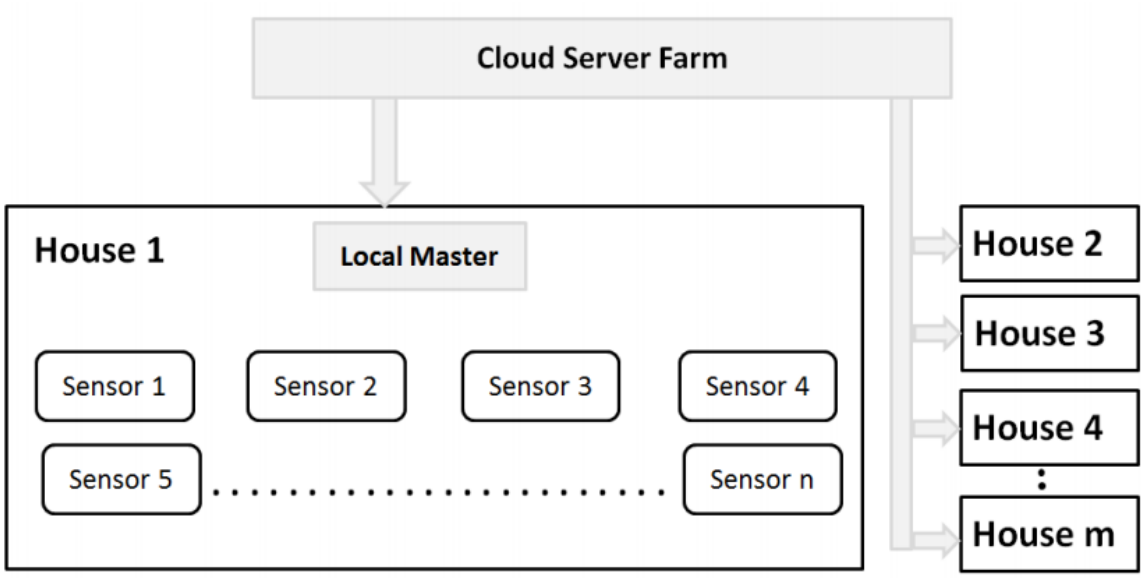
Tabela 2 – Revisão bibliográfica indicando a quantidade de resultados por palavras chave (fonte própria).

47 artigos foram pré-selecionados devido à sua relevância e os seus resumos foram lidos. Destes, 12 foram selecionados para uma leitura completa e usados como base teórica. Dentre estes, foram selecionados 5 artigos mais correlatos com base na relevância nas áreas relacionadas às palavras chave deste trabalho, e serão brevemente sintetizados a seguir.

3.1 POWER EFFICIENT, BANDWIDTH OPTIMIZED AND FAULT TOLERANT SENSOR MANAGEMENT FOR IOT IN SMART HOME

Choubey et al. (2015) propoem um framework para tomada de decisões em uma rede IoT aplicada a smart homes. Conforme representado na figura 9, é considerada a utilização de múltiplos nós sensores IoT em diferentes ambientes de uma casa, onde são utilizadas técnicas de IA nos dados coletados para detectar se existem dependências entre os mesmos. Com base nessas dependências, valores de um conjunto de sensores podem ser previstos com base nos demais conjuntos de sensores; e, com estes conhecimentos, os sensores são ajustados em tempo real visando minimizar redundância e consumo de energia; e em casos de falhas de sensores, decisões aproximadas podem ser tomadas com base nos padrões anteriormente observados.

Figura 9 – Arquitetura proposta. (CHOUBEY et al., 2015)

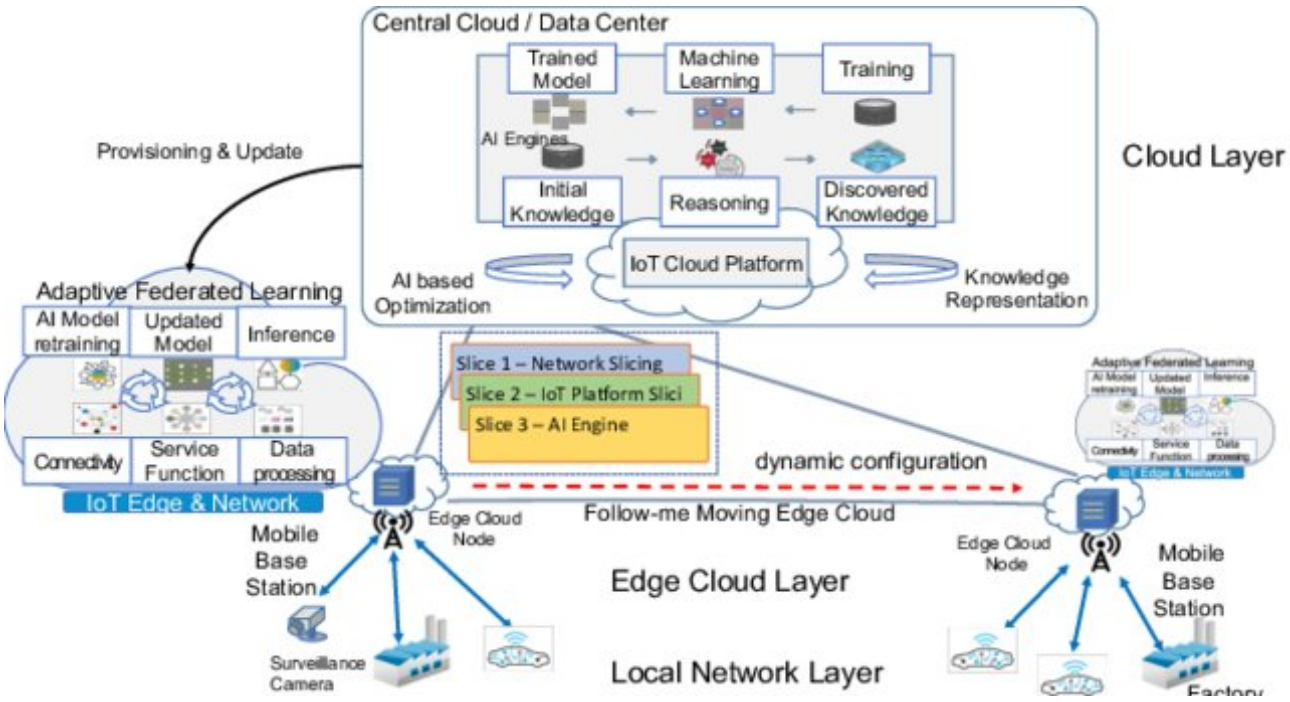


3.2 EIF: TOWARD AN ELASTIC IOT FOG FRAMEWORK FOR AI SERVICES

Fog ainda enfrenta desafios em situações que requerem sensibilidade ao contexto e tomada de decisões automatizadas em tempo real, motivos pelos quais grande parte das plataformas IoT não satisfazem os requisitos da indústria. Com estes desafios em mente, An et al. (2019) propõem o framework EiF (*Elastic Intelligent Fog*), cuja arquitetura geral é representada na figura 10. Esta plataforma utiliza redes neurais profundas para filtrar dados desnecessários de forma a tomar decisões em tempo real: os nós fog possuem uma IA que, ao gerenciar sensores e nós inteligentes adjacentes, oferece aos usuários serviços IoT como fluxo inteligente de tráfego. IA também é utilizada para coordenar fog e nuvem procurando prever falhas e degradação de QoS com base nos dados de monitoramento da rede, reconfigurando estes serviços dinamicamente, com o objetivo de melhorar QoS. O processamento de dados ocorre nas *edges*, procurando alcançar respostas em milisegundos. EiF utiliza raciocínio dedutivo e indutivo para extrair informação de texto desestruturado, podendo compreender linguagem natural ao

complementar texto com um repositório de conhecimento global que pode ser extraído do contexto.

Figura 10 – Arquitetura geral do framework EiF. (AN et al., 2019)



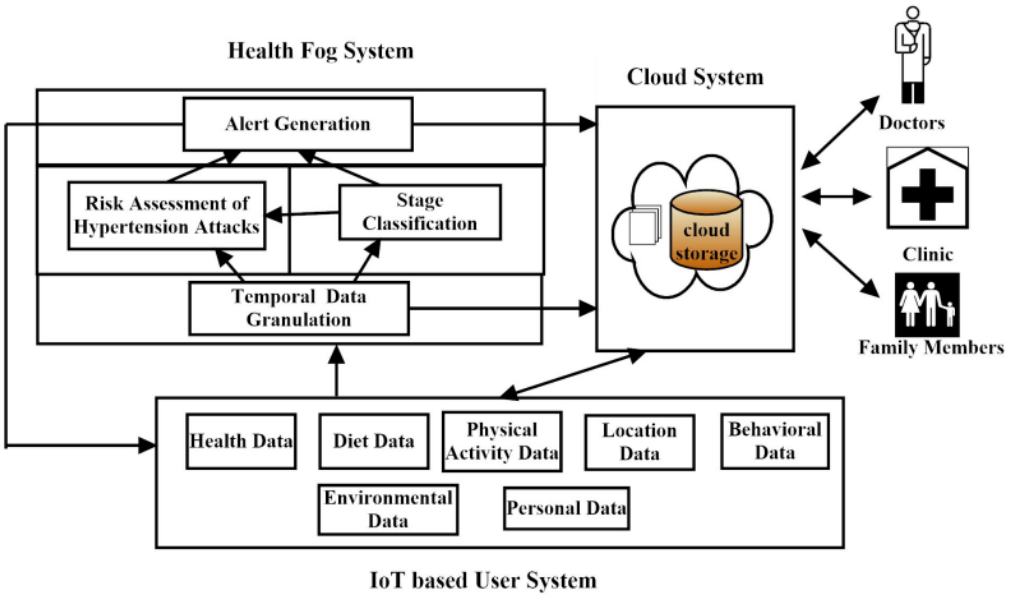
3.3 IOT-FOG BASED HEALTHCARE FRAMEWORK TO IDENTIFY AND CONTROL HYPERTENSION ATTACK

Neste artigo, Sood e Mahajan (2019) propõem um framework IoT-Fog de assistência médica que tem como objetivo identificar crises de hipertensão com base em sintomas do paciente, e notificar emergências aos usuários e médicos em tempo real. Atualmente, para controlar o risco de hipertensão, pacientes devem monitorar certos parâmetros manualmente, visitando centros de saúde; porém este processo pode ser automatizado por meio deste framework no conforto dos seus lares, melhorando a qualidade de vida do paciente. Esta abordagem também pode ser usada para identificar o estágio de hipertensão do paciente por meio de sensores IoT médicos. Como representado na figura 11, este sistema consiste em três subsistemas:

- subsistema de IoT, composto de dispositivos e sensores IoT que capturam os sintomas e os transferem à fog.
- subsistema de fog com *smart gateways*, onde ocorre o processamento e diagnóstico em tempo real. Envia alertas aos telefones celulares dos usuários, possibilitando que medidas de precaução possam ser tomadas a tempo em caso de emergência.
- subsistema de nuvem, onde são armazenados os resultados das análises e históricos médicos dos pacientes, facilitando o acesso aos mesmos por parte de médicos, pacientes e demais usuários do sistema.

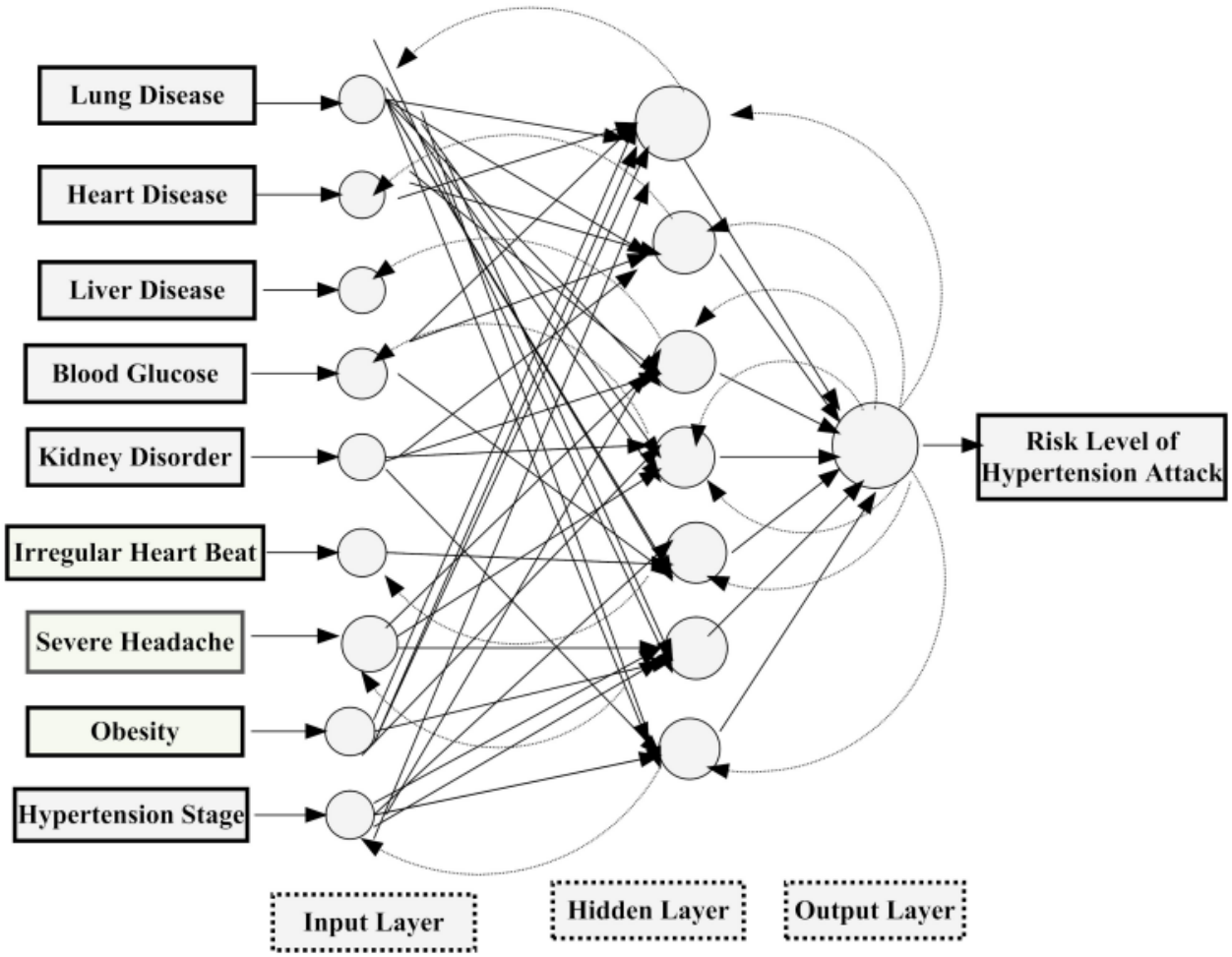
Uma rede neural é utilizada na fog para prever o risco do paciente sofrer uma crise de hipertensão. Como representado na figura 12, este risco é calculado com base no estágio de hipertensão e considerando doenças pulmonares, cardíacas, hepáticas e renais, além do nível de glucose no sangue, batimentos cardíacos, dores de cabeça e obesidade. Cada um dos parâmetros

Figura 11 – Framework do sistema proposto. (SOOD; MAHAJAN, 2019)



anteriores é representado em uma escala entre 0 e 1. Esta rede é treinada por meio de conjuntos de dados onde tanto as entradas quanto as saídas são conhecidas.

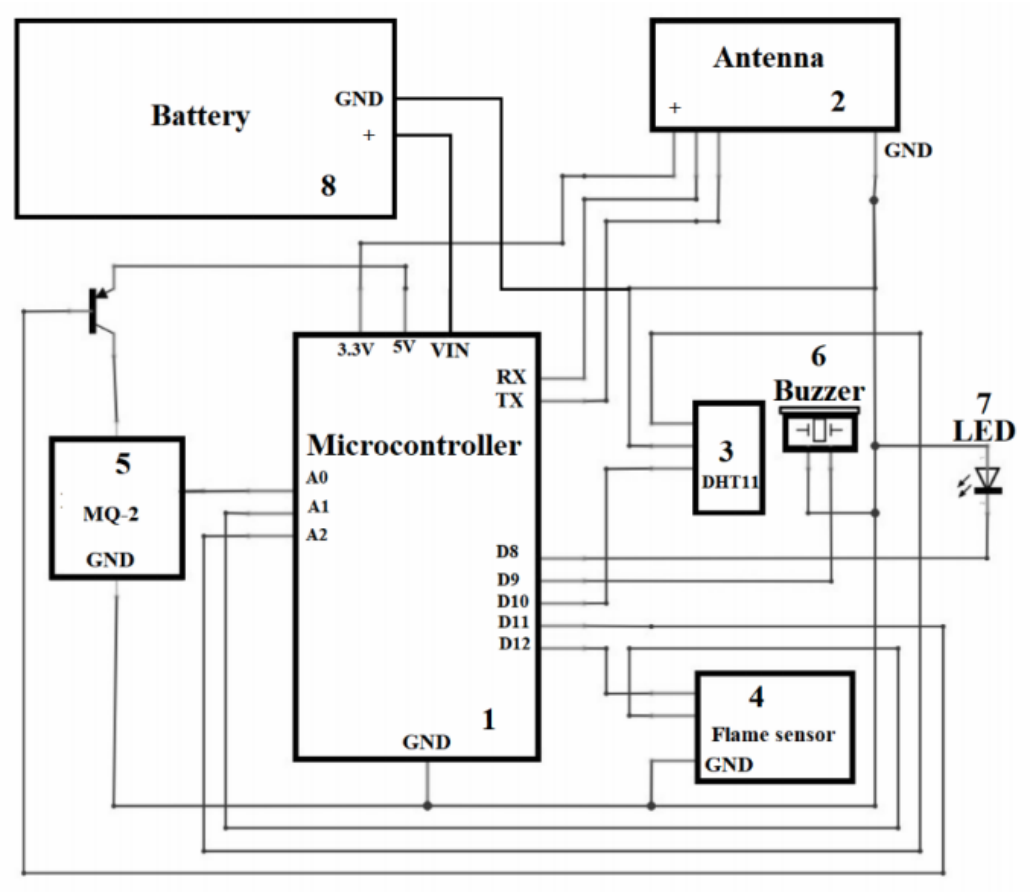
Figura 12 – Rede neural com *backpropagation* utilizada para prever crises de hipertensão. (SOOD; MAHAJAN, 2019)



3.4 AUTONOMIC IOT BATTERY MANAGEMENT WITH FOG COMPUTING

Neste artigo, Sampaio et al. (2019) propõem um sistema de alarme de incêndio para uma casa inteligente, que consiste em um dispositivo IoT e um servidor fog. O dispositivo IoT, ilustrado na figura 13, mede a temperatura e umidade do ar, a concentração de gás, fumaça e presença de chamas; e envia estas condições do ambiente à fog, onde estes dados serão processados, armazenados e exibidos ao usuário por meio de um website. Foi criada uma tabela de estados, onde para cada sensor foi definido um valor de limiar superior que indica a alteração de estado. Foi exemplificado considerando os seguintes valores como limites dos sensores: 35 graus para o sensor de ar; 4% para concentração de gases e presença ou não de chamas. Quando todos os sensores excedem seus respectivas limiares, é considerado um estado de emergência e uma sirene e uma luz LED são acionados. Os dados dos sensores foram capturados de forma independente entre si, por meio de threads em um pipeline, com o objetivo de economizar energia. Durante cada ciclo, o dispositivo IoT estará acionado durante o tempo de resposta dos sensores, que é de 2 segundos, e logo, se uma emergência não for detectada, entrará no modo de economia de energia até o próximo ciclo. Estimativas de tempo de vida da bateria são apresentadas, tendo em vista o tempo máximo de resposta permitido por lei, que é de 5 segundos. Ao utilizar uma bateria de 2000 mA, este dispositivo tem autonomia estimada de até 18 horas contínuas, e até 45 horas com um ciclo de *sleep* de 3 segundos.

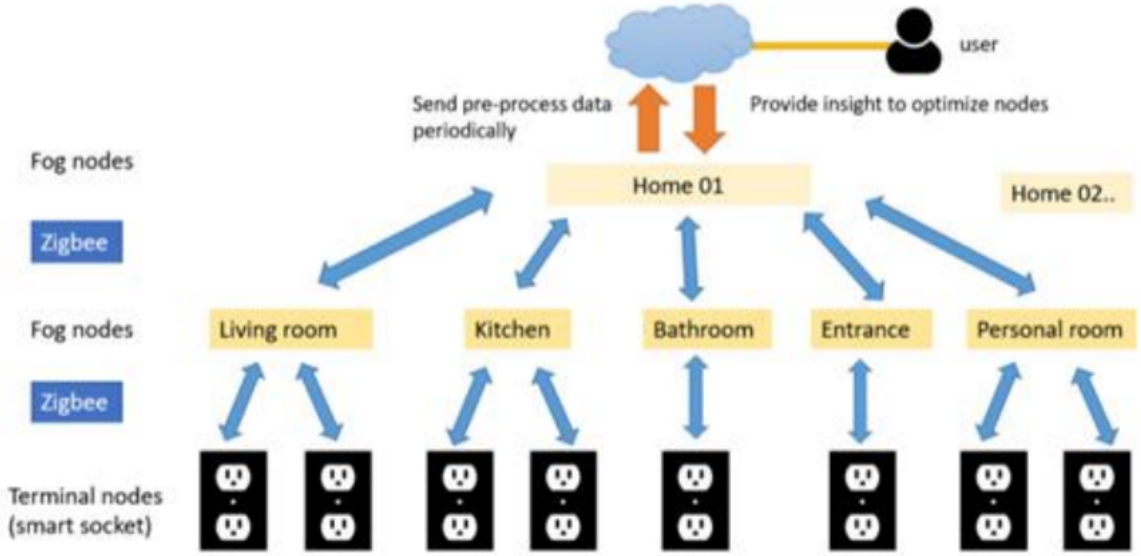
Figura 13 – Visão geral do dispositivo IoT proposto. (SAMPAIO et al., 2019)



3.5 DESIGN AND IMPLEMENTATION OF A POWER CONSUMPTION MANAGEMENT SYSTEM FOR SMART HOME OVER FOG-CLOUD COMPUTING

Neste artigo, Chen, Azhari e Leu (2018) propõem um modelo de gerenciamento de energia para uma casa inteligente por meio de uma rede fog. Conforme representado na figura 14, a arquitetura proposta consiste na nuvem e três camadas de fog: a primeira camada (camada base) possui vários nodos IoT (tomadas inteligentes), cada nó da segunda camada representa um quarto da casa inteligente, e os nós da terceira camada representam a casa em si. Cada camada de fog é independente das demais, e o seu poder computacional e de armazenamento é proporcional à camada em que se encontra; e a comunicação entre camadas ocorre por meio do protocolo *Zigbee*. As tomadas inteligentes são dispositivos IoT que têm como objetivo medir o consumo energético dos equipamentos conectados à mesma e enviá-los à camada base. A nuvem recebe os dados de uma a quatro vezes por mês, e com eles o usuário pode decidir se o consumo de algum nó deverá ser ajustado de forma a reduzir o consumo de energia da casa. Quando comparado com uma abordagem baseada puramente na nuvem, o framework proposto apresentou tempo de processamento significativamente menor, melhor desempenho e alta eficiência.

Figura 14 – Visão geral do modelo proposto. (CHEN; AZHARI; LEU, 2018)

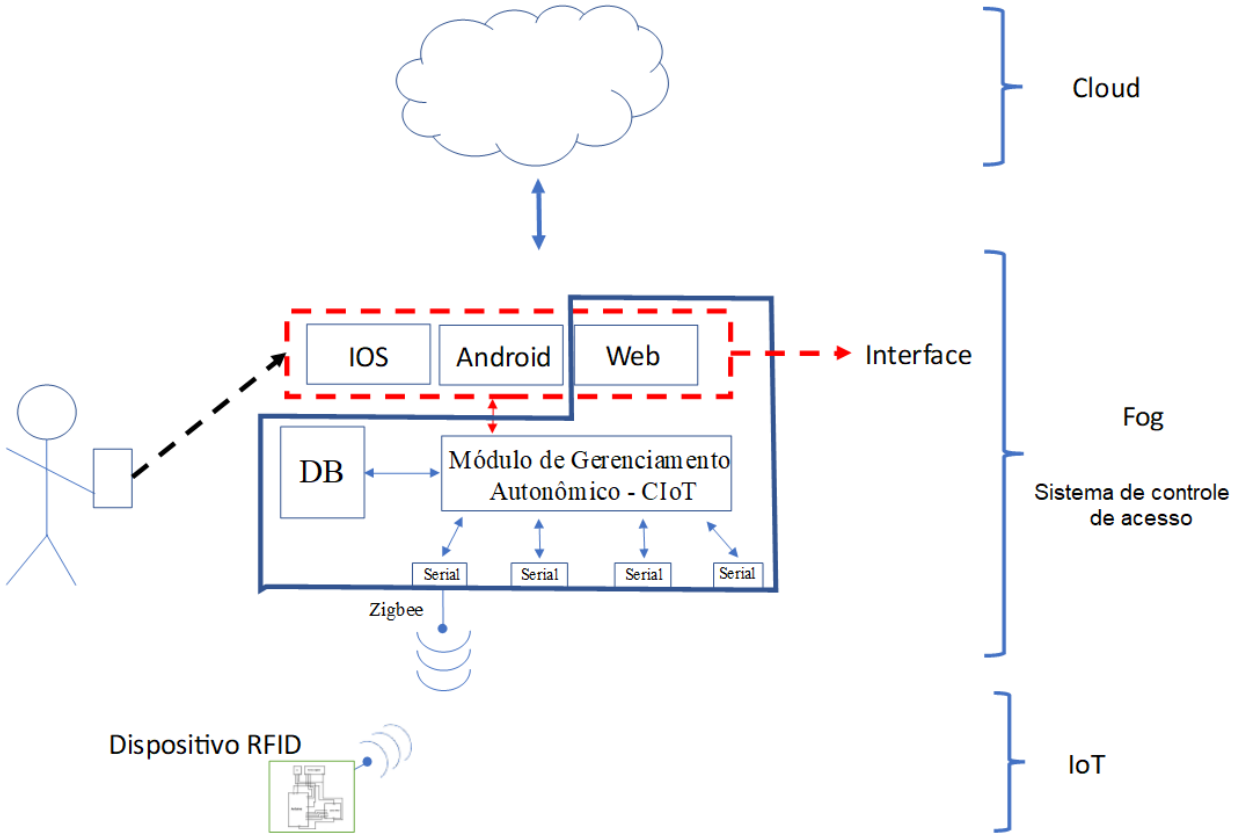


4 DESENVOLVIMENTO DA PROPOSTA

Como proposta, este trabalho considera o contexto de gerenciamento de energia em condomínios inteligentes com o desenvolvimento de um sistema de controle de acesso, assim como o trabalho de gerenciamento de energia de dispositivos IoT do sistema de detecção de incêndio proposto por Sampaio et al. (2019).

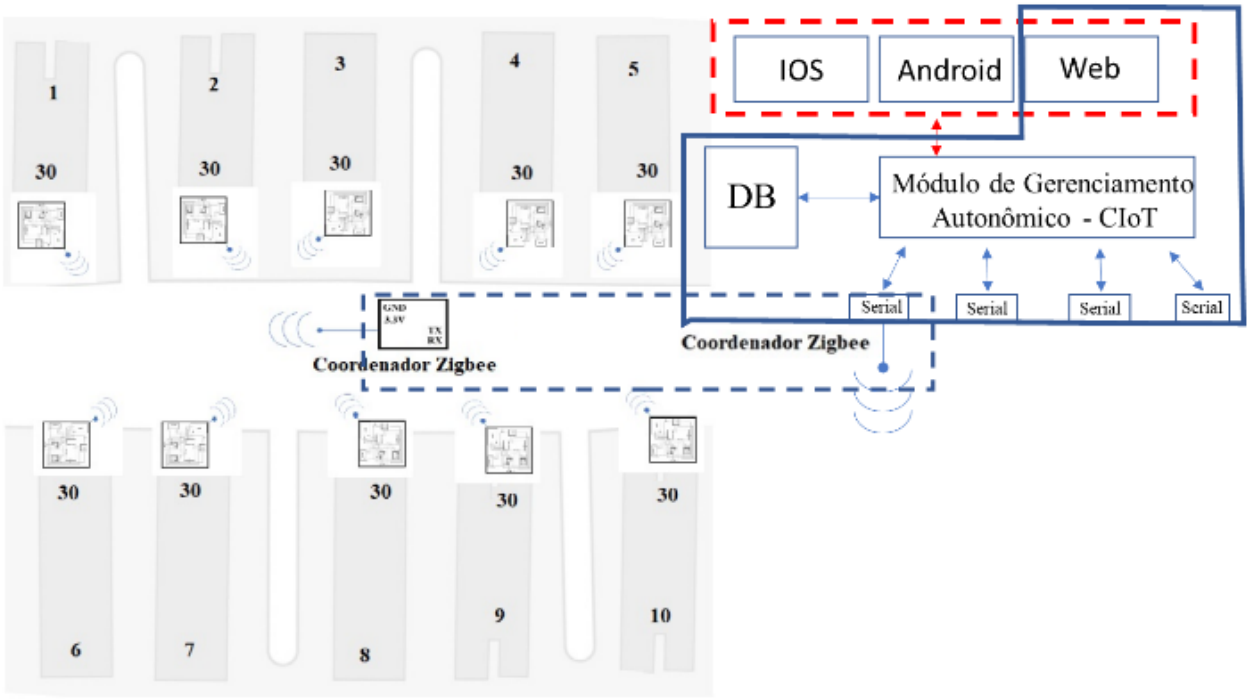
A partir dos dados de histórico de acesso, capturados por meio de um sistema de RFID, propomos um sistema inteligente, utilizando a técnica de redes neurais, que tem como objetivo determinar automaticamente os horários em que os apartamentos do condomínio estão desocupados, o que permitiria aumentar o tempo que os dispositivos IoT, de detecção de incêndio, permaneçam em modo de economia de energia e, desta forma, reduzir o consumo (medido em kWh) e o custo (medido em reais). Uma visão geral do modelo proposto é apresentada na figura 15.

Figura 15 – Visão geral do modelo proposto (fonte própria).



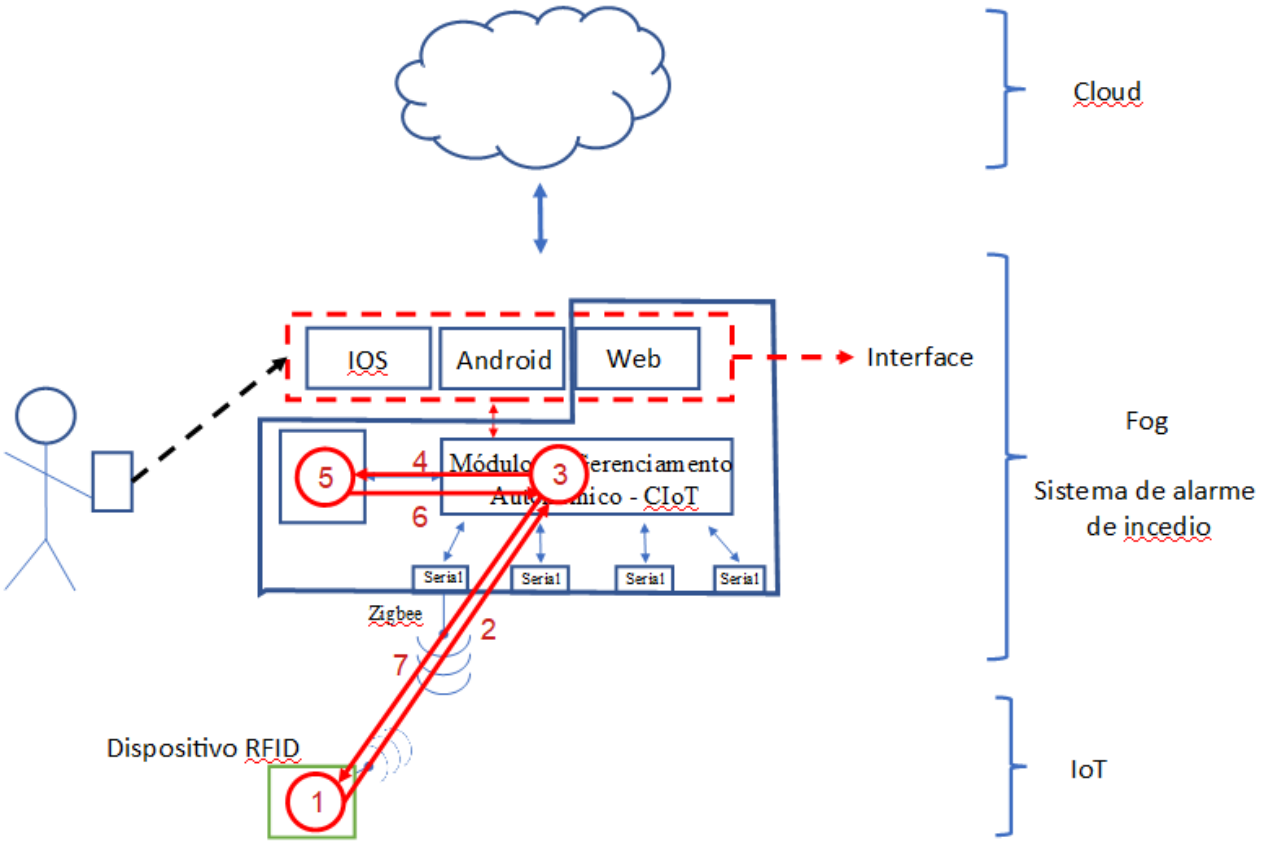
A figura 16 ilustra o condomínio considerado, que consiste em 10 edifícios, cada um com 30 apartamentos; onde cada apartamento possui um sensor RFID e um nó de detecção de incêndio, totalizando 600 nós. Os dados são enviados por meio do protocolo *Zigbee*, a um nó fog central para processamento na rede neural.

Figura 16 – Visão geral do condomínio inteligente considerado para a proposta (fonte própria).



A figura 17 ilustra o fluxo geral das mensagens no sistema. Os passos indicados são detalhados a seguir:

Figura 17 – Fluxo geral das mensagens no sistema proposto (fonte própria).



- 1: O dispositivo IoT detecta a presença de um cartão RFID e lê o seu valor.

- 2: O valor do cartão RFID é encapsulado em um pacote Zigbee e é enviado ao nó fog.
- 3: O nó fog recebe o pacote zigbee, extrai o valor do cartão RFID e o horário em que este pacote foi recebido.
- 4: Estes dois dados são enviados para o banco de dados.
- 5: Os dados são armazenados e é verificado se o cartão é autorizado para acessar a residência em questão.
- 6: É enviada uma mensagem informando se o cartão possui autorização de acesso ou não.
- 7: Esta informação é repassada para o dispositivo IoT em questão, o qual ativa o relê permitindo o acesso à residência, caso autorizado, ou mostra uma mensagem de erro ao usuário, caso contrário.

Para este trabalho, foi considerado apenas um habitante por apartamento, e os dados iniciais serão obtidos por meio de simulações. Dentre as entradas consideradas pelo sistema podemos mencionar o identificador de cada apartamento, dia e horário. A saída deste sistema será uma variável que indica se a residência estará desocupada nesse momento. O treinamento da rede neural ocorrerá na nuvem e a rede treinada será implementada na rede fog.

4.1 DESENVOLVIMENTO DO PROTÓTIPO DE NÓ IOT

É proposto um protótipo do dispositivo IoT com RFID, que é composto por um Arduino Uno, uma antena Zigbee, uma antena RFID e um relê. Este protótipo irá ler o cartão RFID do usuário, registrar o acesso no banco de dados e acionar o relê, liberando a porta para que o usuário possa entrar na sua residência.

Também é proposto um dispositivo fog composto por um Raspberry Pi e uma antena Zigbee, que recebe os pacotes enviados pelo dispositivo IoT e os armazena em um banco de dados, e executa os algoritmos de inteligência artificial.

A rede neural utiliza os dados de histórico de acesso salvos pelo protótipo no banco de dados para inferir os horários em que o usuário estará ou não em sua residência. Esta informação pode ser usada para ajustar o tempo de *sleep* de outros dispositivos IoT, otimizando o consumo energético.

4.1.1 Desenvolvimento do nó final

Para implementar o dispositivo final, foi utilizada uma placa Arduino Uno, uma antena XBee Zigbee, uma antena RFID MFRC522 e um relê.

4.1.2 Protocolo *Zigbee*

De acordo com Chen, Azhari e Leu (2018), o protocolo *Zigbee* tem como base o padrão IEEE 802.15.4, que define a operação de dispositivos com baixa velocidade de transmissão e baixo consumo energético, quando comparado com outros protocolos utilizados no meio acadêmico e comercial, como Wi-Fi e Bluetooth. De acordo com ZigBee Alliance (2014), dispositivos *Zigbee* podem cumprir os seguintes três papéis:

- Coordenador: cria a rede e configura aspectos como o canal e PAN ID; e é possível utilizar múltiplas redes diferentes em um único local físico. Nesta configuração, os dispositivos não entram em modo de economia de energia (*sleep*).

- Roteador: tem como função distribuir o tráfego na rede. Assim como na configuração coordenador, os dispositivos nesta configuração não entram em modo *sleep*; mas podem armazenar em cache alguns dados referentes aos seus nós filhos, quando estes estiverem em modo *sleep*.

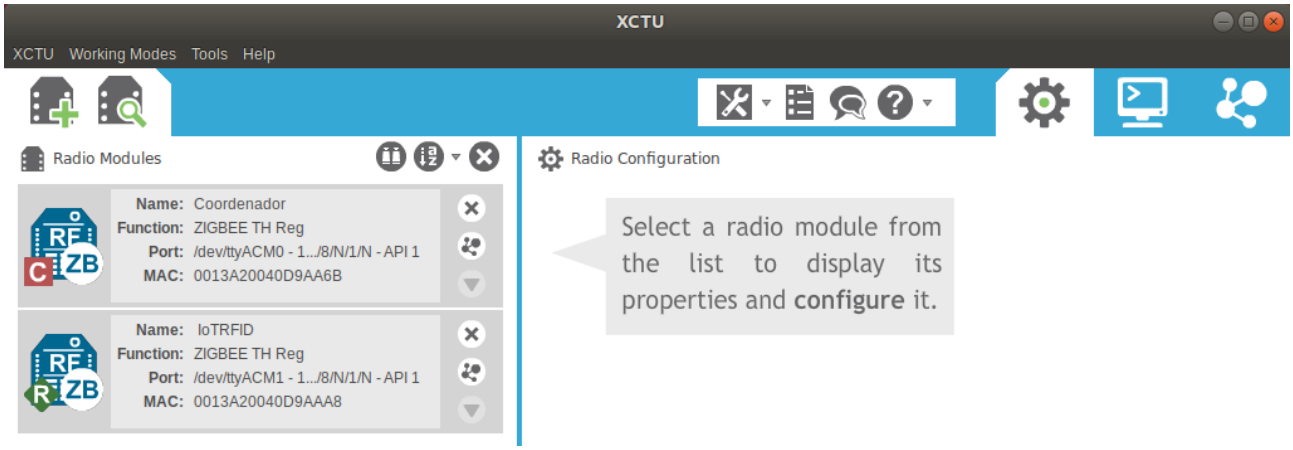
- Dispositivos finais: geralmente possuem sensores, e têm como finalidade capturar os dados lidos por tais sensores e transmiti-los a um nó roteador ou coordenador. Estes dispositivos podem entrar em modo de economia de energia periodicamente.

Para o protótipo de nó IoT, foi escolhida a antena *Xbee Zigbee*, fabricada pela empresa Digi, e suporta o protocolo *Zigbee*. Cada nó permite até 14 nós filhos, tem alcance de 60 metros, frequência de 2.4 GHz e velocidade de transmissão de 250 Kbps.

Para configurar a antena *Xbee*, foi utilizado o aplicativo XCTU, desenvolvido pela Digi. Esta ferramenta permite alterar o papel da antena, PAN ID, endereço de destino dos pacotes, entre outros.

A configuração do nó coordenador e do dispositivo IoT é apresentada na figura 18 no software XCTU. O nó coordenador é identificado pelo quadrado vermelho com um "C"e o dispositivo final, configurado no modo roteador, é identificado por um losango verde com um "R".

Figura 18 – Interface do XCTU com ambos dispositivos e suas respectivas configurações (fonte própria).



4.2 DESENVOLVIMENTO DO SISTEMA FOG

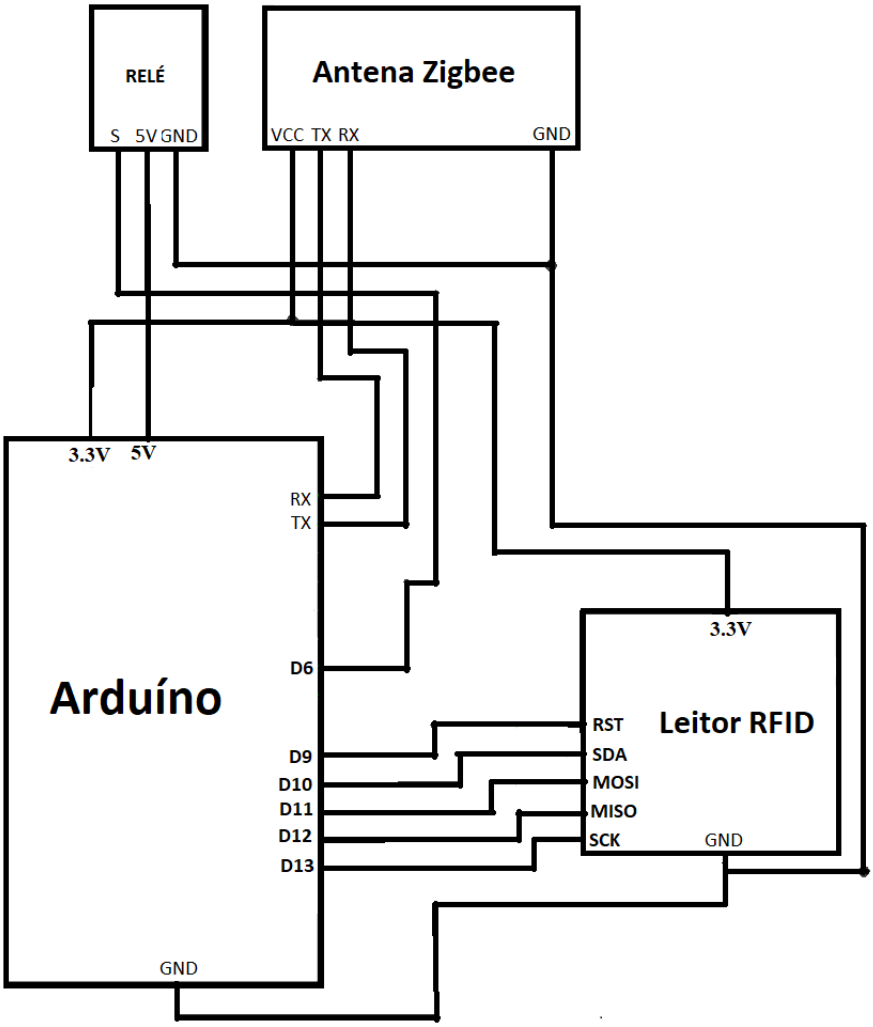
O nó coordenador possui a antena Xbee e a placa Raspberry Pi. As portas de transmissão (TX) e recebimento (RX) de ambos componentes foram conectadas, além das ligações de energia (3.3 V) e *ground* (GND), conforme demonstrado na figura 19.

A porta serial tanto da antena quanto do Arduino foram configuradas com a taxa de transmissão de 115200 bits/seg. De acordo com Sampaio e Motoyama (2017b), a antena Xbee possui dois modos de funcionamento, configuráveis pelo software XCTU:

- Modo AT (Transparente): este modo é utilizado para enviar dados de um dispositivo final para um coordenador. O coordenador, ao receber estes pacotes pela porta serial RX, os encapsula automaticamente e os envia para o endereço de destino conforme configurado no software XCTU. Para enviar estes dados, basta utilizar o comando *Serial.print()*.

- Modo API (*Application programming interface*): este modo é utilizado para enviar dados para múltiplos destinatários. É necessário montar os pacotes manualmente, byte a byte, por meio do comando *Serial.write()*.

Figura 19 – Conexões entre o Arduíno e os demais componentes do dispositivo final (fonte própria).



4.2.1 Montagem de pacotes

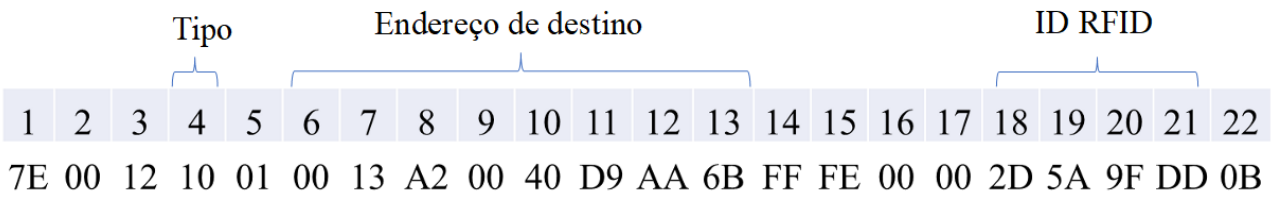
O dispositivo final foi configurado no modo AT e a biblioteca MFRC522, disponibilizada pelo fabricante da antena RFID, foi utilizada para capturar os valores dos tags RFID, conforme ilustrado no algoritmo 1. Este módulo possui um microcontrolador próprio que lê o ID do cartão, representado por 4 bytes, e o envia para o microcontrolador do Arduíno por meio das portas MISO e MOSI. Ao receber este dado, o microcontrolador do Arduíno monta o pacote e o envia pela sua porta TX para a porta RX da antena Zigbee, que então transmite este pacote pela rede (SAMPAIO; MOTOYAMA, 2017a).Os pacotes enviados, chamados de "pacotes de requisição de transmissão", têm tamanho fixo de 22 bytes neste exemplo.

```
1 for (int i = 0; i < 4; i++) {
2     if (mfrc522.uid.uidByte[i]<16) Serial.print("0");
3     Serial.print(mfrc522.uid.uidByte[i], HEX);
4     }
```

Algoritmo 1 – Captura do valor do tag RFID (fonte própria).

Sempre que a antena RFID ler um tag, um pacote será montado e enviado ao Raspberry Pi. Na linha 3 do algoritmo 1, o valor da tag, representado por quatro bytes, é adicionado ao pacote e enviado à antena, que finaliza a montagem do pacote Zigbee adicionando as demais informações que foram pré-configuradas, incluindo o endereço de destino. Ao receber o valor da tag, a antena XBee o transforma em valores hexadecimais, byte a byte; logo estes dados serão encapsulados em um pacote e enviados ao Raspberry Pi. A figura 20 ilustra o pacote criado quando a antena RFID captura o valor *2D 5A 9F DD*.

Figura 20 – Exemplo de pacote de requisição de transmissão (fonte própria).



Como o pacote Zigbee foi configurado para ter tamanho fixo, é importante tomar o cuidado de adicionar um zero à esquerda se o valor lido for menor do que 16 (ou 0x10, em hexadecimal), a fim de garantir que dois caracteres sejam sempre enviados por vez.

Ao receber o pacote Zigbee na fog, ele é processado e é verificado se o ID RFID possui permissão de acesso para aquela residência. Em caso positivo, a fog envia um pacote de confirmação para o dispositivo IoT, o qual aciona o relê durante dois segundos. Para este exemplo foi considerado que ao ser acionado o relê, a porta da residência será aberta, permitindo o acesso à residência; e uma vez que tal porta seja fechada, será automaticamente trancada.

A figura 21 descreve detalhadamente a estrutura do pacote da figura 20. É importante notar que os campos de ID do pacote, endereço de destino (tanto de 64 bits quanto de 16 bits), raio de *broadcast* e opções foram configurados anteriormente por meio do software XCTU. Os campos de tamanho do pacote, tipo do pacote e *checksum* são populados automaticamente pelo microcontrolador da antena.

Figura 21 – Estrutura do pacote de requisição de transmissão (fonte própria).

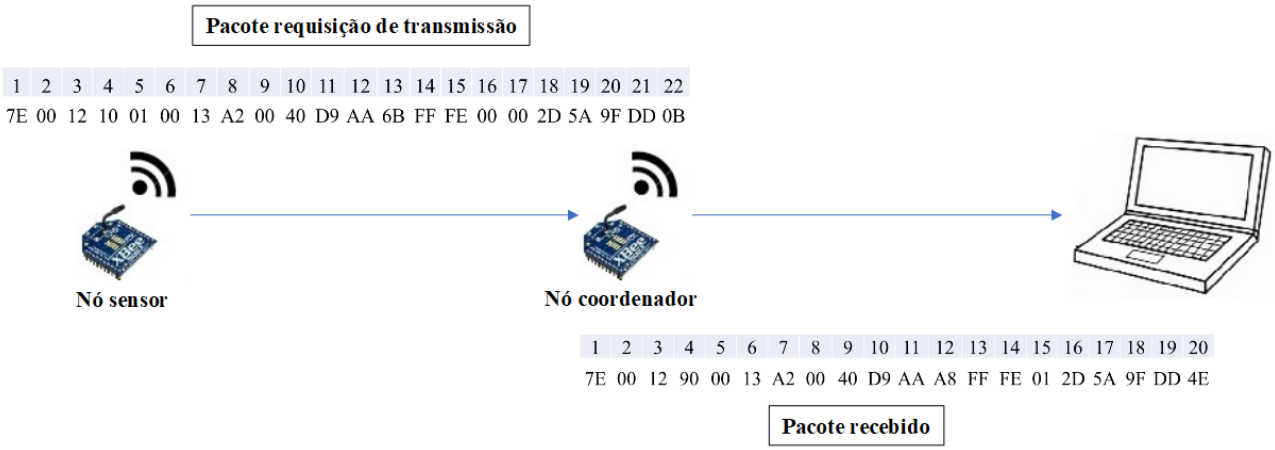
Byte	Descrição	Valor hexadecimal	Significado
1	Delimitador inicial	7E	
2 e 3	Tamanho do pacote	00 15	18 bytes de tamanho
4	Tipo do pacote	10	Requisição de transmissão
5	ID do pacote	01	Identificação do pacote
6 a 13	Endereço de destino de 64-bit	00 13 A2 00 40 D9 AA 6B	Endereço do nó coordenador
14 e 15	Endereço de destino de 16-bit	FF FE	Envio para todos roteadores
16	Raio de broadcast	00	Pulos máximos de broadcast
17	Opções	00	
18 a 21	Dados	2D 5A 9F DD	Valor ID cartão RFID
22	Checksum	0B	

De acordo com Sampaio e Motoyama (2017a), quando a antena XBee é configurada no modo API, os pacotes devem ser montados de forma manual seguindo o padrão do tipo de pacote, já que, ao contrário do modo AT, o encapsulamento não é realizado de forma automática. Esta característica oferece mais liberdade para modificar o pacote.

De acordo com Sampaio e Motoyama (2017b), ao receber o pacote, a antena XBee acoplada ao Raspberry Pi o processa, substituindo o endereço de destino pelo endereço do remetente e excluindo os campos de raio de *broadcast* e ID do frame. Logo, o pacote modificado é enviado pela porta RX à porta TX do Raspberry Pi.

A figura 22 representa a transmissão do pacote da figura 20. Assim que a antena XBee destino recebe este pacote, ela o processa e o envia pela porta serial, permitindo obter posteriormente os dados capturados pela antena RFID.

Figura 22 – Transmissão e processamento do pacote da figura 20 (fonte própria).



A figura 23 ilustra detalhadamente a estrutura do pacote processado, denominado como pacote de recebimento.

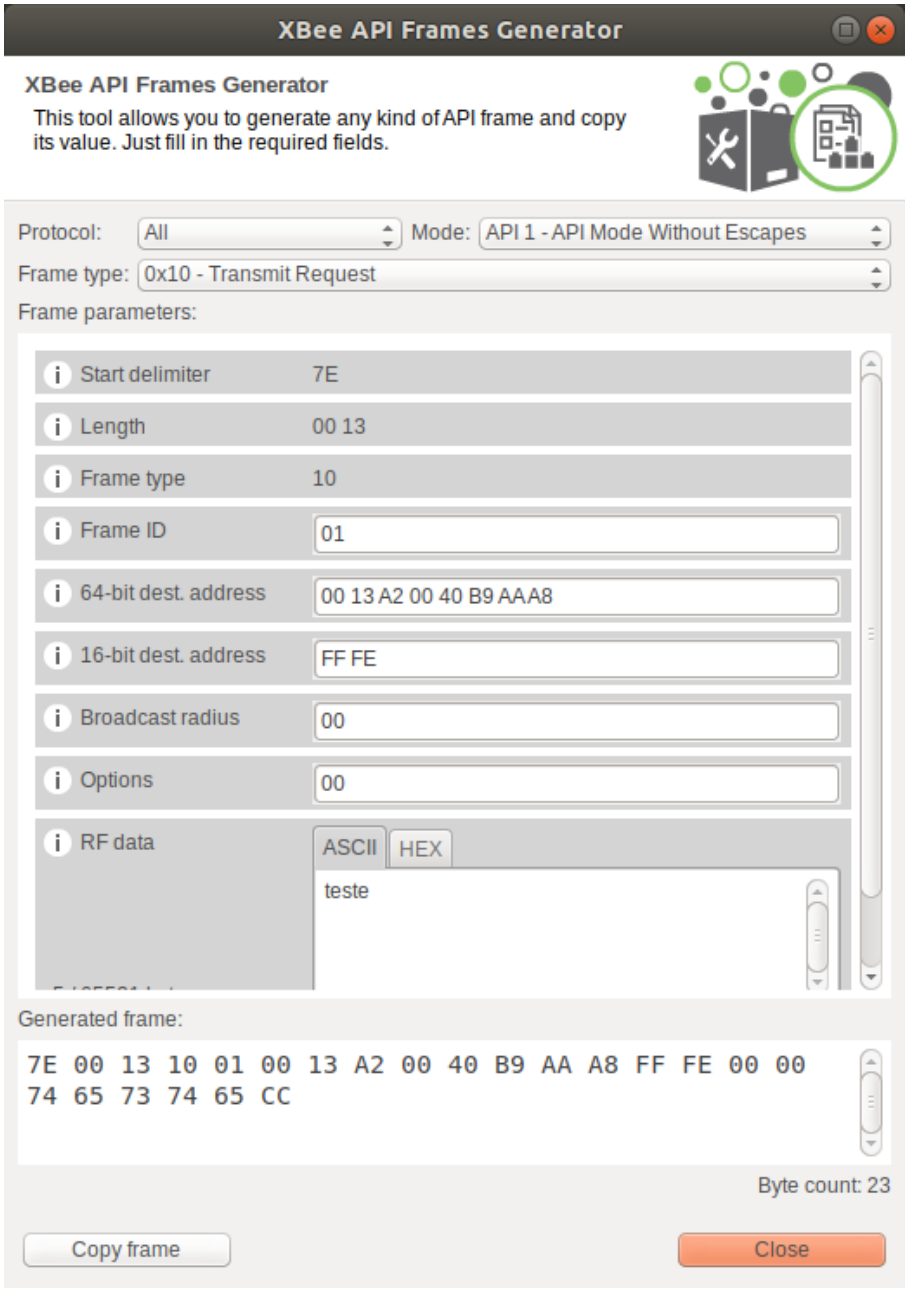
Figura 23 – Estrutura do pacote de recebimento (fonte própria).

Byte	Descrição	Valor hexadecimal	Significado
1	Delimitador inicial	7E	
2 e 3	Tamanho do pacote	00 15	21 bytes de tamanho
4	Tipo do pacote	90	Pacote recebido
5 a 12	Endereço de origem de 64-bit	00 13 A2 00 40 D9 AA A8	Endereço do nó IoT
13 e 14	Endereço de origem de 16-bit	FF FE	Endereço desconhecido
15	Opções	00	
16 a 19	Dados	2D 5A 9F DD	Valor ID cartão RFID
20	Checksum	B1	

A ferramenta *XBee frame generator*, disponível como parte do software XCTU, pode auxiliar na criação manual de pacotes. Esta ferramenta apresenta a estrutura de diversos tipos de pacotes, facilitando a criação dos mesmos e ajudando a evitar erros. A figura 24 ilustra um exemplo de uso desta ferramenta.

Para este trabalho, a antena do Arduino foi configurada no modo AT, e a do Raspberry Pi no modo API.

Figura 24 – Interface da ferramenta XCTU, com um exemplo de pacote de transmissão (fonte própria).



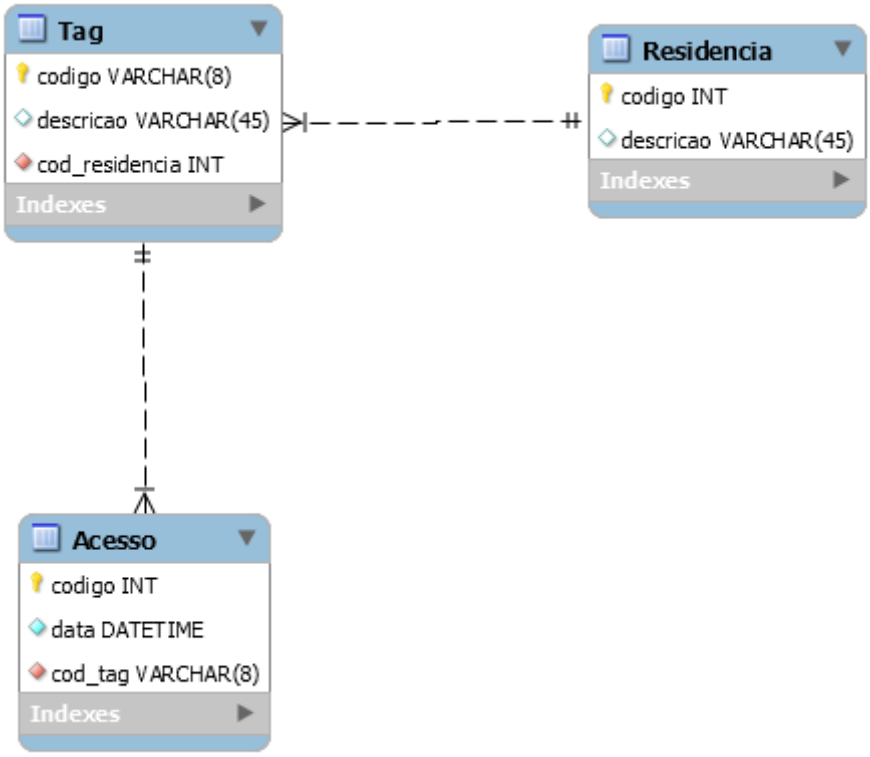
5 GERENCIAMENTO AUTONÔMICO COM IA

O dispositivo IoT proposto, ao ler um tag RFID, envia o valor deste tag para o nó fog. Este valor é recebido por um servidor, escrito na linguagem de programação Python, que envia este dado e o horário de recebimento para um banco de dados MySQL. Neste capítulo é descrito o modelo de armazenamento de dados, a simulação e tratamento de dados, e o desenvolvimento da rede neural.

5.1 ARMAZENAMENTO DE DADOS

Para armazenar os dados gerados pelo dispositivo IoT, foi escolhido o banco de dados relacional MySQL. Este banco de dados possui três tabelas, conforme ilustrado na figura 25.

Figura 25 – Esquema do banco de dados MySQL (fonte própria).



- Residência: representa cada uma das residências do condomínio. Possui um código serial, isto é, um inteiro com incremento automático que identifica unicamente cada residência; assim como um campo para armazenar uma breve descrição da residência.
- Tag: representa cada uma das tags RFID do condomínio. A sua chave primária é o próprio código do tag, que é lido pela antena RFID do dispositivo final. Também possui um campo para adicionar uma breve descrição da tag ou do dono da mesma, e o identificador único da residência à qual este tag pertence; portanto, cada tag somente pode pertencer a uma residência e possuir um único dono.
- Acesso: esta tabela armazena os acessos às residências. A sua chave primária é um inteiro serial, o atributo *data* armazena a data e hora em que o acesso ocorreu, e o atributo *cod_tag* é o código da tag RFID referente ao acesso.

A priori, devem ser cadastradas residências e tags no banco de dados por meio do terminal MySQL ou por outra ferramenta de gerência de banco de dados. Os acessos são armazenados automaticamente pelo servidor python. É interessante notar que a data e hora são atribuídos pelo servidor e não pelo dispositivo final, reduzindo desta forma o tamanho dos pacotes e a largura de banda requerida.

5.2 SIMULAÇÃO DE DADOS

A fim de treinar a rede neural, foram gerados dados por meio da ferramenta *Libreoffice Calc*. Esta ferramenta permite gerar dados que seguem uma distribuição estatística, como beta, binomial, qui-quadrado, exponencial, gamma, hipergeométrica, lognormal, normal, uniforme e Weibull, entre outras.

Esta ferramenta foi escolhida por permitir a manipulação de dados de forma simples e intuitiva, permitindo utilizar formatos como *datetime*, compatível com SQL; assim como por oferecer suporte a várias funções lógico-matemáticas, como somatório, piso e módulo.

Conforme ilustrado na tabela 3, foram considerados cinco cenários que representam os perfis de grupos de moradores, conforme os horários de entradas e saídas dos mesmos.

Tabela 3 – Cenários simulados, com os seus respectivos cronogramas e quantidade de residências (fonte própria).

Cenário	1	2	3	4	5
Quantidade de residências	150	40	30	60	20
Primeiro acesso (saída)	08:00:00	06:00:00	11:00:00	7:00:00	17:00:00
Segundo acesso (entrada)	11:30:00	13:00:00	16:00:00	9:30:00	02:00:00
Terceiro acesso (saída)	12:30:00		20:00:00	13:00:00	
Quarto acesso (entrada)	18:00:00		22:00:00	15:40:00	
Quinto acesso (saída)				18:00:00	
Sexto acesso (entrada)				20:50:00	

Para cada cenário foram simulados valores de entradas e saídas diários, considerando apenas dias úteis semanais. Dessa forma, para representar um mês foram gerados dados para 22 dias uteis.

Para gerar os valores dos horários de acesso, considera-se uma distribuição normal com a média sendo o valor do acesso representado na tabela 3 e com desvio padrão de 15 minutos.

Assume-se que antes do primeiro acesso, a pessoa está em sua residência, logo o primeiro acesso corresponde a uma saída; o segundo acesso representa o morador retornando à sua residência; o terceiro uma nova saída, e assim por diante.

Ao seguir a configuração de números no padrão brasileiro, a separação decimal corresponde a uma vírgula. Esta característica ocasionou dois problemas durante o desenvolvimento das simulações:

- Como o formato CSV também utiliza a vírgula por padrão para separar campos da tabela, a estrutura da tabela era corrompida. Para resolver este problema, o separador de campos do arquivo CSV foi alterado para um ponto e vírgula (;).
- Posteriormente, ao importar os dados salvos neste padrão para a rede neural, estes números eram erroneamente identificados como *string*, pois Python segue o formato americano, onde a separação decimal é simbolizada por um ponto. Para tratar este problema, a configuração foi alterada para o padrão americano nos campos que serão utilizados pela rede neural.

Também é importante destacar que foram encontrados problemas causados por erros de arredondamento numéricos, portanto, funções de comparação foram reformuladas de forma a utilizar operadores de comparação, como maior e menor, em lugar do operador de igualdade.

Após criar a simulação, estes dados foram exportados no formato CSV. Este formato armazena os dados de uma tabela em um arquivo de texto, separando-os por vírgula, permitindo a manipulação dos mesmos por outro programa, de forma simples.

5.2.1 Tratamento de dados

As entradas e saídas da rede neural devem ser representadas em formato numérico. De acordo com Kuźniar e Zając (2015), o método de preprocessamento de dados é um fator essencial para determinar o sucesso de uma aplicação de redes neurais. Com este objetivo, após executar a simulação, os dados foram codificados no formato numérico *float*, no software *Libreoffice Calc*, e logo armazenados em formato CSV, para a sua posterior aplicação na rede neural. A codificação utilizada para as variáveis é a seguinte:

- A data foi representada como a quantidade de dias decorridos desde primeiro de janeiro de 1900.
- A hora, incluindo minutos e segundos, foi codificada como um número real, pertencente ao conjunto $[0, 1)$.
- O tag foi representado como um número decimal positivo.
- Para representar se a residência está desocupada ou não, foi utilizada uma variável que assume o valor 1 quando desocupada e 0 quando não.

5.3 REDE NEURAL

Para implementar a rede neural, foi utilizado o API Keras, que pode ser considerado como um *wrapper* para a linguagem de programação de redes neurais de baixo nível TensorFlow. A rede em si foi descrita na linguagem Python. Neste capítulo é apresentada uma breve introdução ao Keras e ao Tensorflow, assim como uma descrição detalhada da implementação da rede neural.

5.3.1 Desenvolvimento da rede neural

O algoritmo 2 descreve a rede neural. É importante destacar que as funções utilizadas são implementadas pelo Keras, que por sua vez utiliza o Tensorflow. Este algoritmo é detalhado, passo a passo, a seguir:

```
1  dataset = genfromtxt(r'../simulacao/csv/cenarios_1_a_6.csv',
   ↪  encoding='latin-1', delimiter=',', skip_header=2,
2                       usecols=(2, 3, 4, 5))
3  X = dataset[:, 0:3]
4  Y = dataset[:, 3]
5
6  model = Sequential()
7  model.add(Dense(12, input_dim=3, activation='relu'))
8  model.add(Dense(24, activation='relu'))
9  model.add(Dense(1, activation='sigmoid'))
10
11 model.compile(loss='mean_squared_error', optimizer='sgd',
   ↪  metrics=['accuracy'])
12
13 model.fit(X, Y, epochs=30, batch_size=10)
14 _, accuracy = model.evaluate(X, Y)
```

Algoritmo 2 – Código da rede neural em Python com Keras (fonte própria).

- A linha 1 representa a importação dos dados do arquivo CSV. Conforme detalhado na seção 5.2.1, os dados estão armazenados neste arquivo em formato numérico.
- A linha 3 separa as primeiras três colunas (0, 1 e 2) do arquivo CSV para servirem como entrada da rede neural. Estas colunas representam o dia, hora e tag.
- A linha 4 separa a coluna 3 para servir como a saída da rede neural. Este dado corresponde a uma variável que indica se a residência está desocupada no momento, conforme descrito na seção 5.2.1.
- A linha 6 indica que o modelo da rede neural é sequencial.
- A linha 7 descreve a camada de entrada. O primeiro parâmetro é a quantidade de neurônios desta camada, o segundo indica que a rede possui três entradas, e o último indica o modo de ativação da rede, neste caso, unidade linear retificada.
- A linha 8 representa a camada intermediária ou oculta, com 24 neurônios e como modo de ativação a unidade linear retificada. Como esta rede é rasa, ela possui uma única camada intermediária.
- A linha 9 representa a camada de saída. Possui um único neurônio, que corresponde à saída da rede e o modo de ativação é sigmoidal.
- A linha 11 indica como o modelo será compilado. O primeiro argumento é a função de perda, neste caso, erro quadrático médio. O segundo argumento indica o otimizador, neste caso gradiente descendente estocástico. O último parâmetro representa a função de métrica, isto é, a função que mede o desempenho da rede; neste caso, foi escolhida a acurácia.
- A linha 13 indica como o modelo será treinado. O primeiro argumento corresponde aos dados de entrada, o segundo aos de saída, o terceiro indica a quantidade de épocas e o último indica o tamanho de lote, ou *batch*.
- A linha 14 avalia o treinamento da rede com respeito à acurácia.

A rede neural realiza a tarefa de classificação, pois dado um dia, horário e tag, a rede irá determinar se a residência está desocupada (valor 1) ou ocupada (valor 0).

Com o objetivo de encontrar uma configuração que apresentasse uma melhor acurácia, foram testadas diversas variações dos seguintes parâmetros: número de camadas, quantidade de neurônios de cada camada, modos de ativação, funções de perda e otimizadores. Alguns desdes testes e as suas respectivas acurácias são apresentados na tabela 4.

Os parâmetros da solução escolhida correspondem aos apresentados na primeira linha da tabela 4, assim como na figura 26. Nesta figura, a entrada é composta por três dados, e o indicador *units* representa a quantidade de neurônios da camada. Após o treinamento da rede neural com os dados descritos na seção 5.2, a precisão obtida da rede foi de 65,65%.

A figura 27 apresenta a acurácia da rede neural escolhida em relação à quantidade de épocas. Como podemos ver, a acurácia se mantém estável após a quinta época, logo uma quantidade maior de épocas é desnecessária. A figura 28 apresenta a taxa de perda representada por meio do erro quadrático médio. Como podemos ver, o erro fica aproximadamente estável a partir da quinta época, reforçando a conclusão obtida pela figura 27. Com estas informações, a quantidade de épocas escolhida para treinar a rede neural foi de 10 épocas.

Tabela 4 – Alguns modelos de redes neurais com as suas respectivas acurácias (fonte própria).

Camada 1	Camada 2	Camada 3	Camada 4	Acurácia (%)
12	24	1		65,65
12	60	1		34,35
12	12	1		34,35
12	5	1		34,35
12	24	24	1	65,65
12	60	60	1	34,35
12	5	5	1	34,35

Figura 26 – Modelo da rede neural desenvolvida (fonte própria).

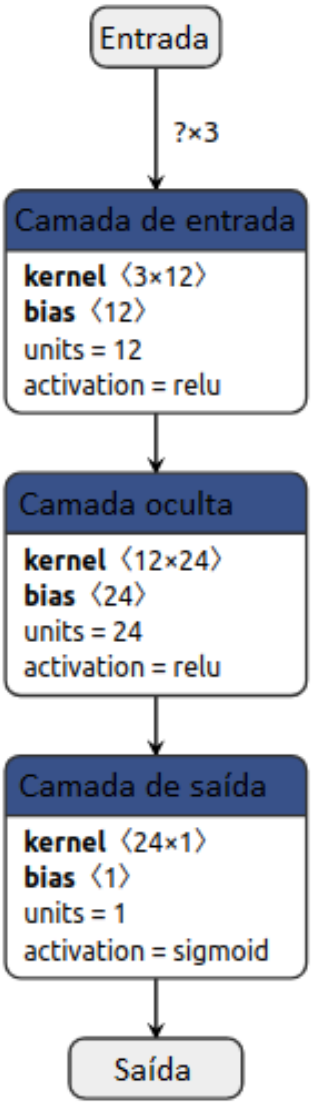


Figura 27 – Acurácia obtida pela rede neural em relação à quantidade de épocas (fonte própria).

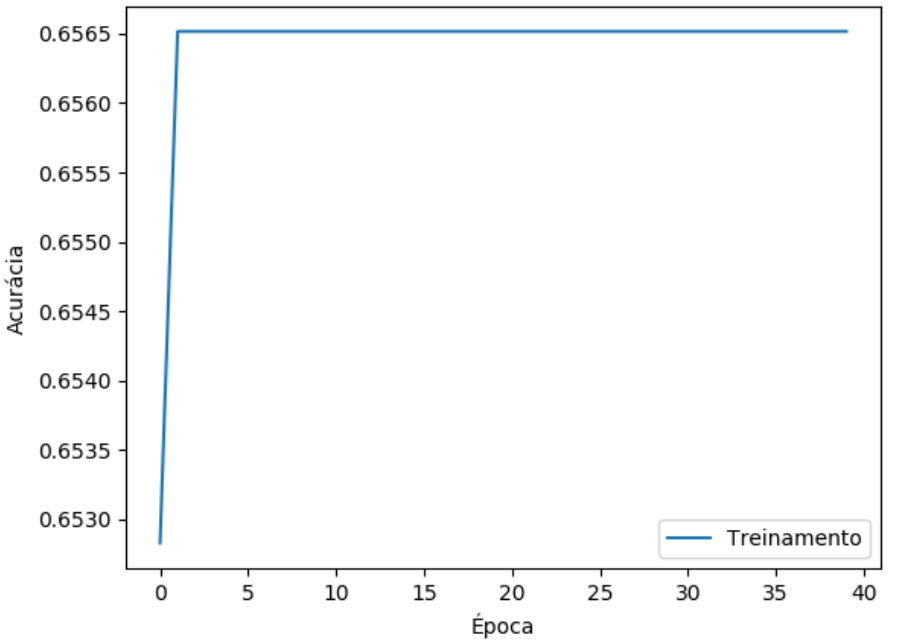
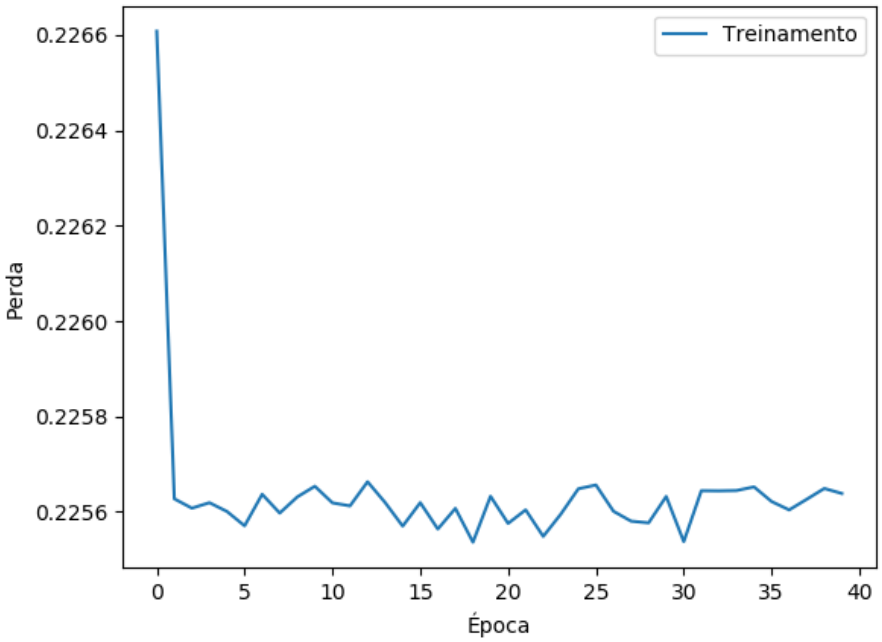


Figura 28 – Perda da rede neural em relação à quantidade de épocas (fonte própria).



6 CONCLUSÕES E TRABALHOS FUTUROS

Para este trabalho, foi utilizada uma rede neural com três camadas, e a precisão obtida foi de 65,65%. Esta precisão poderia ser melhorada ao realizar uma análise mais detalhada de redes neurais, utilizar mais dados para o treinamento, e possívelmente ao utilizar redes neurais profundas ou convolucionais.

Uma otimização adotada com o objetivo de economizar banda é que a data e hora são definidas pelo servidor, e não pelo dispositivo IoT. Desta forma, o único dado enviado pela rede é o tag RFID.

O conhecimento gerado pela rede pode ser utilizado para diversas aplicações, possibilitando aumentar a qualidade de vida dos moradores e diminuir desperdícios; assim como possibilitar o gerenciamento autônomo de energia e outros recursos.

Dentre os trabalhos futuros, podemos citar:

- Utilizar redes neurais profundas ou convolucionais e procurar obter melhor acurácia na predição de dados.
- Além dos parâmetros utilizados neste trabalho, considerar também outras variáveis que possam alterar a rotina dos moradores, como alguns sensores meteorológicos, de entrada para a rede neural. Também podem ser consideradas outras condições, como feriados, dias letivos, finais de semana, etc.
- Com o fim de obter dados mais relevantes para o desenvolvimento do trabalho, utilizar dados reais, ou simulações mais sofisticadas, para o treinamento da rede neural.
- Integrar completamente o banco de dados à rede neural.
- Desenvolver uma interface web para oferecer os demais serviços no banco de dados, como cadastrar residências, tags, habitantes, e outros.
- Expandir este trabalho para suportar múltiplos habitantes e múltiplos tags por residência.
- Utilizar as informações obtidas pela rede neural para realizar o gerenciamento autônomo da residência.
- Utilizar estes conhecimentos para realizar o gerenciamento autônomo de energia, procurando reduzir desperdícios.
- Utilizar estes conhecimentos para otimizar a quantidade de dados gerados e armazenados no banco de dados.
- Estes conhecimentos também podem ser utilizados para uma infinidade de aplicações que melhorem a qualidade de vida das pessoas. Por exemplo, acionar o sistema de ar condicionado em um dia quente, alguns minutos antes da chegada de um morador, de forma a oferecer uma temperatura ambiente agradável imediatamente após a chegada do mesmo.

REFERÊNCIAS

ABADI, M. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. <https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45166.pdf>.

ALRAWAIS, A. et al. Fog computing for the internet of things: Security and privacy issues. *IEEE Internet Computing*, v. 21, p. 34–42, 2017.

AN, J. et al. Eif: Toward an elastic iot fog framework for ai services. *IEEE Communications Magazine*, v. 57, n. 5, p. 28–33, 05 2019. <https://www.researchgate.net/publication/333069163>.

BUYYA, R. et al. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, Elsevier B.V., v. 25, n. 6, p. 599–616, 2009. ISSN 0167739X. <http://dx.doi.org/10.1016/j.future.2008.12.001>.

CHEN, Y. D.; AZHARI, M. Z.; LEU, J. S. Design and implementation of a power consumption management system for smart home over fog-cloud computing. *IGBSG 2018 - 2018 International Conference on Intelligent Green Building and Smart Grid*, IEEE, p. 1–5, 2018.

CHOLLET, F. et al. *Keras documentation*. 2015. <https://keras.io/>.

CHOUBEY, P. K. et al. Power efficient, bandwidth optimized and fault tolerant sensor management for iot in smart home. *Souvenir of the 2015 IEEE International Advance Computing Conference, IACC 2015*, p. 366–370, 2015. <https://www.researchgate.net/publication/283125637>.

DINH, H. T. et al. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, v. 13, n. 18, p. 1587–1611, 2013. <https://onlinelibrary.wiley.com/doi/abs/10.1002/wcm.1203>.

DOSILOVIC, F. K.; BRCIC, M.; HLUPIC, N. Explainable artificial intelligence: A survey. *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2018 - Proceedings*, p. 210–215, 2018. <https://www.researchgate.net/publication/325398586>.

GUPTA, H. et al. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Software - Practice and Experience*, v. 47, n. 9, p. 1275–1296, 2017. ISSN 1097024X.

IORGA, M.; MARTIN, M. J.; FELDMAN, L. Fog computing conceptual model nist special publication 500-325. p. 11, 2018. <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-325.pdf>.

JENNINGS, B.; STADLER, R. Resource Management in Clouds: Survey and Research Challenges. *Journal of Network and Systems Management*, v. 23, n. 3, p. 567–619, 2015. ISSN 10647570. <https://doi.org/10.1007/s10922-014-9307-7>.

KUźNIAR, K.; ZAJąC, M. Some methods of pre-processing input data for neural networks. *Computer Assisted Methods in Engineering and Science*, v. 22, p. 141–151, 2015.

LUGER, G. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. University of New Mexico, New Mexico, United States: Pearson Education Inc, 2009. 754 p. ISSN 1895-1767. ISBN 9780321545893.

RAY, P. P. A survey on Internet of Things architectures. *Journal of King Saud University - Computer and Information Sciences*, King Saud University, v. 30, n. 3, p. 291–319, 2018. ISSN 22131248. <https://doi.org/10.1016/j.jksuci.2016.10.003>.

SAMPAIO, H.; MOTOYAMA, S. Implementation of a greenhouse monitoring system using hierarchical wireless sensor network. *2017 IEEE 9th Latin-American Conference on Communications, LATINCOM 2017*, v. 2017-January, p. 1–6, 2017.

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, p. 89–103, 2019. ISSN 16113349.

SOOD, S. K.; MAHAJAN, I. Iot-fog-based healthcare framework to identify and control hypertension attack. *IEEE Internet of Things Journal*, v. 6, n. 2, p. 1920–1927, April 2019. ISSN 2327-4662.

YASUMOTO, K.; YAMAGUCHI, H.; SHIGENO, H. Survey of Real-time Processing Technologies of IoT Data Streams. *Journal of Information Processing*, v. 24, n. 2, p. 195–202, 2016.

ZigBee Alliance. *ZigBee-PRO Stack Profile: Platform restrictions for compliant platform testing and interoperability*. Dezembro de 2014. <https://zigbee.org/download/standard-zigbee-pro-specification/>.

Controle de acesso em ambientes inteligentes com Fog e redes neurais

René Nolio Santa Cruz[1]

[1]Departamento de informática e estatística – Universidade Federal de Santa Catarina (UFSC)
Florianópolis, SC, Brazil

santacruzrene@gmail.com

Abstract. *Advances in computing turned the cloud into a commodity where resources can be rented based on demand. IoT networks emerge to integrate devices (or "things") into the Internet, but have little storage space and can generate large amounts of data. IoT networks have been integrated into the cloud in order to take advantage of its elastic properties; creating the fog paradigm, which seeks to preprocess data before sending it to the cloud. On the other hand, AI techniques have proven to be efficient in several areas, especially for data classification and prediction. It's proposed a model of neural networks in fog that, with the access data in a smart condominium, the times when the homes are unoccupied will be estimated. This knowledge can be used for a variety of optimizations and to improve residents' quality of life. The viability of this model is demonstrated by a fog network prototype.*

Resumo. *Avanços na área de computação têm transformado a nuvem num commodity, onde recursos podem ser alugados com base na demanda. Redes IoT surgem visando integrar dispositivos (ou "coisas") à Internet, porém, possuindo pouco espaço de armazenamento e podendo gerar grande quantidade de dados. Redes IoT foram integradas à nuvem com o objetivo de usufruir sobre as propriedades elásticas da mesma; dando origem ao paradigma de fog, que procura pré-processar os dados antes de enviá-los à nuvem. Por outro lado, técnicas de IA se mostraram eficientes em diversas áreas, em especial para classificação e predição de dados. É proposto um modelo de redes neurais na fog que, a partir dos dados de acesso em um condomínio inteligente, são estimados os horários em que as residências se encontram desocupadas. Este conhecimento pode ser utilizado para uma infinidade de otimizações e para melhorar a qualidade de vida dos moradores. A viabilidade deste modelo é demonstrada por meio de um protótipo de rede fog.*

1. Introdução

O paradigma de IoT procura integrar coisas à Internet, visando criar um mundo de dispositivos conectados (GUPTA et al., 2017). De acordo com Ray (2018) e Gupta et al. (2017), as coisas, ou *things*, são dispositivos que possuem interfaces de comunicação com e sem fio, são munidos de sensores, e tipicamente possuem baixo poder computacional e baixa capacidade de armazenamento; porém podem produzir quantidades imensas de dados. Exemplos de coisas incluem sensores inteligentes, dispositivos médicos, geladeiras, veículos, câmeras inteligentes e outros (GUPTA et al., 2017).

A computação na nuvem é um paradigma que consiste em compartilhar recursos computacionais por meio da Internet, geralmente com base na demanda, como um serviço ou *commodity* (JENNINGS; STADLER, 2015). De acordo com Iorga, Martin e Feldman (2018), foram propostas diversas arquiteturas que integram os paradigmas de nuvem e IoT. Uma destas abordagens consiste em transmitir os dados gerados pelos dispositivos IoT à nuvem, para o seu posterior processamento e armazenamento. Tendo em vista que, de acordo com Iorga, Martin e Feldman (2018), a quantidade de dispositivos inteligentes interconectados deverá passar de 50 bilhões até o ano 2020, esta abordagem se torna inviável, especialmente para aplicativos de tempo real.

Dentre os diversos desafios em aberto na área de IoT, podemos destacar a grande quantidade de dados heterogêneos gerados que, ao serem transmitidos para a nuvem, geram grande consumo de banda, alta latência e baixa qualidade de serviço (CHOUBEY et al., 2015). Em particular, a nuvem é eficiente para processar dados em *batch*, mas não é ideal para processamento em tempo real (JENNINGS; STADLER, 2015). Para amenizar estas desvantagens, foi proposto o paradigma de *fog computing*, que consiste em utilizar dispositivos de uma camada intermediária entre IoT e nuvem para pré-processar os dados.

Por outro lado, técnicas de inteligência artificial também podem ser utilizadas para amenizar os mesmos problemas. Choubey et al. (2015) utilizam redes neurais para encontrar relações entre os valores medidos por sensores IoT, com o objetivo de reduzir a redundância de sensores e minimizar o consumo energético. Sood e Mahajan (2019) propõem um *framework* que, por meio de redes neurais, estima o risco de ataque de hipertensão a partir de dados capturados por meio de sensores IoT, permitindo notificar serviços de emergência a tempo. An et al. (2019) propõem um framework que utilizam algoritmos adaptativos distribuídos de IA para gerenciar nodos fog e coordená-los com a nuvem, com o objetivo de melhorar o QoS.

Desta forma, procurou-se combinar o paradigma de *fog computing* e técnicas de IA para encontrar os horários em que as residências de um condomínio estão desocupadas.

1.1. Objetivo geral

O objetivo principal é desenvolver um dispositivo IoT para gerenciar o controle de acesso de apartamentos inteligentes, assim como utilizar técnicas de inteligência artificial para encontrar os horários em que a residência está desocupada. Estes conhecimentos permitiriam aplicar várias otimizações que aumentam a qualidade de vida dos moradores e reduz desperdícios. É importante destacar que este trabalho não tem como objetivo conduzir uma análise detalhada na área de inteligência artificial, mas sim aplicar as técnicas e conhecimentos existentes em uma rede fog-IoT.

1.2. Objetivos específicos

Os objetivos específicos deste trabalho são:

- Propor e desenvolver um dispositivo IoT com um sensor RFID e um ambiente fog para processar os valores medidos.
- Utilizar técnicas de IA para encontrar os horários em que a residência está desocupada conforme o perfil de diferentes residências.
- Validar este projeto por meio de testes, com um protótipo de rede fog, assim como avaliar a análise dos dados, obtidos por meio de simulações, gerada pela rede neural.

1.3. Justificativa

Dentre os principais problemas encontrados em redes fog podemos destacar o grande volume de dados gerados pelos dispositivos IoT, que devem ser transmitidos pela rede. Por outro lado, técnicas de IA demonstraram ser eficientes em diversas áreas, e em especial, para lidar com problemas de classificação e predição de dados.

Ao aplicar técnicas de IA na fog, é possível inferir horários nos quais as residências estão desocupadas, o que permitiria o gerenciamento autonômico de sistemas IoT da residência, por exemplo, otimizar o consumo energético dos dispositivos IoT ao otimizar o tempo de *sleep* dos mesmos.

1.4. Método de pesquisa e trabalho

Para o desenvolvimento deste trabalho de conclusão de curso, os principais métodos de pesquisa e trabalho são:

1. Pesquisa de livros e artigos recentes (preferencialmente a partir de 2015) nas áreas de IoT, IA, fog e demais áreas relevantes, no contexto de ambientes inteligentes (casas, estufas, ambientes inteligentes).
2. Análise detalhada dos artigos mais correlatos e focados nesta proposta, de modo a compreender corretamente os conceitos sobre diversos pontos de vista.
3. Validação da solução proposta através de um protótipo, utilizando dados obtidos por meio de simulações.
4. Implementação do protótipo proposto, realização de testes e análise dos resultados obtidos.

1.5. Organização deste artigo

Este relatório está organizado da seguinte maneira: Na seção 1 é apresentada a introdução; as subseções 1.1 e 1.2 detalham os objetivos gerais e específicos deste trabalho, respectivamente; a subseção 1.3 apresenta a justificativa para o desenvolvimento deste trabalho e a subseção 1.4 descreve o método de pesquisa e trabalho seguido.

A seção 2 apresenta os conceitos fundamentais utilizados como base teórica, como cloud, fog, IoT, inteligência artificial e redes neurais. Na seção 3, foram apresentados os resultados da revisão bibliográfica e 5 artigos correlatos foram detalhados.

A seção 4 apresenta a proposta. A subseção 4.1 descreve como o protótipo de nó IoT foi desenvolvido, o protocolo utilizado e o sistema fog. A seção 4.2.1 descreve o desenvolvimento do sistema de gerenciamento com IA, como os dados foram armazenados, como foram simulados, e o desenvolvimento da rede neural utilizada.

A seção 5 cita as conclusões e trabalhos futuros que podem ser desenvolvidos com base neste trabalho. Finalmente, são descritas as referências bibliográficas.

2. Conceitos Fundamentais

Neste capítulo são apresentados brevemente alguns conceitos fundamentais necessários à confecção deste trabalho e as principais técnicas e tecnologias já conhecidas e utilizadas nos meios acadêmico e comercial, nas áreas de *cloud*, fog e IoT, além de uma breve introdução à inteligência artificial e à área de redes neurais.

2.1. Cloud Computing

Na última década, avanços na computação como um serviço e nas tecnologias de virtualização, permitiram a construção de *data centers* lucrativos em alta escala, que executam grande parte dos aplicativos na Internet e processamento em *backend*, permitindo que a infraestrutura de data centers seja alugada a terceiros. Segundo Jennings e Stadler (2015), emerge, então, o paradigma de Computação em Nuvem (*Cloud Computing*), no qual um conjunto de recursos computacionais é compartilhado entre aplicativos que o acessam pela Internet. Este conceito pode também se referir a hardware e software de sistema que reside nos data centers que hospedam tais aplicativos. Os objetivos dos provedores de nuvem estão centrados no uso eficiente dos recursos, dentro de limites estabelecidos por um Acordo de Nível de Serviço, ou *Service Level Agreement* (SLA), que é um contrato formal entre o provedor e o usuário, cujo objetivo é definir os aspectos funcionais e não funcionais do serviço em termos quantitativos (JENNINGS; STADLER, 2015).

De acordo com Dinh et al. (2013), dependendo do nível de abstração no qual o serviço é oferecido, ambientes de nuvem pública podem ser classificados em três principais categorias:

- IaaS (*Infrastructure-as-a-Service*, ou infraestrutura como serviço): O provedor da nuvem irá prover recursos procurando cumprir com o SLA acordado com o usuário da nuvem. O SLA define formal e quantitativamente aspectos dos serviços oferecidos, como disponibilidade, desempenho, tolerância a falhas, entre outros. O provedor da nuvem pode oferecer diferentes níveis de serviço e priorizar certos aspectos dos serviços dependendo do SLA e condições operacionais. O usuário da nuvem geralmente possui um SLA com os usuários finais e procura explorar a elasticidade de recursos presente na nuvem para acomodar as necessidades dos seus clientes. O custo para o cliente geralmente é baseado na quantidade de recursos utilizados; a infraestrutura pode ser expandida ou encolhida dinamicamente conforme necessário.
- PaaS (*Platform-as-a-Service*, ou plataforma como serviço): Oferece um ambiente integrado para desenvolver, testar e implantar aplicativos. Exemplos deste modelo de serviço incluem *Google App Engine* e *Microsoft Azure*.
- SaaS (*Software-as-a-Service*, ou software como serviço): É capaz de distribuir software com requisitos específicos. Os usuários podem acessar aplicativos remotamente e o custo é baseado na quantidade de recursos utilizados. Exemplos deste modelo de serviço incluem *Salesforce* e *Microsoft's Live Mesh*.

2.2. Internet of Things

O paradigma de IoT tem como objetivo integrar à internet "coisas" (ou *"things"*, em inglês), como dispositivos médicos, geladeiras, câmeras e sensores. Este paradigma permite novas formas de interação entre coisas e pessoas, e permite melhorar a qualidade de vida e a utilização de recursos (GUPTA et al., 2017). Dá origem a uma visão de um mundo de dispositivos e pessoas conectados. De acordo com Gupta et al. (2017), estima-se que até 2025 IoT terá um impacto econômico de 11 trilhões de dólares, e cerca de um trilhão de dispositivos IoT serão implantados. Os desafios mais importantes são heterogeneidade, escalabilidade, interoperabilidade, segurança e privacidade (YASUMOTO;

YAMAGUCHI; SHIGENO, 2016). Para refletir situações reais, processamento em tempo real de dados é requerido; porém, abordagens baseadas em nuvem possuem atrasos consideráveis, diminuindo a qualidade do serviço e desperdiçam recursos da nuvem e banda; e com milhões de dispositivos, esta abordagem é inviável (YASUMOTO; YAMAGUCHI; SHIGENO, 2016),(GUPTA et al., 2017).

As limitações de processamento e armazenamento dos dispositivos de IoT, combinadas com o alto processamento de dispositivos *edge*, deram origem à fog, ou computação em névoa, que estende serviços da nuvem à edge, resultando em redução de latência. Dados passam por vários dispositivos entre o seu ponto de origem e o destino, e é importante utilizar a capacidade computacional e de armazenamento destes dispositivos intermediários (ex: gateways, roteadores, etc) (GUPTA et al., 2017).

2.3. Fog Computing

Fog computing ou computação em névoa (doravante referenciado apenas como fog) pode ser definido como um paradigma computacional distribuído, que estende os serviços oferecidos pela nuvem às *edges* (ou bordas) da rede (GUPTA et al., 2017); Facilita o gerenciamento de serviços de rede, computação e armazenamento entre *data centers* e dispositivos, e foi proposto para diminuir o espaço existente entre data centers e dispositivos IoT (ALRAWAIS et al., 2017).

Fog envolve a execução de aplicativos tanto na nuvem como em dispositivos intermediários entre a nuvem e as coisas, como gateways e roteadores, por exemplo (GUPTA et al., 2017). Fog suporta mobilidade, distribuição geográfica, escalabilidade, heterogeneidade de recursos e interfaces, interação com a nuvem, análise distribuída de dados e baixa latência. Os objetivos principais deste paradigma são reduzir o volume de dados, reduzir o tráfego entre dispositivos IoT e a nuvem, diminuir a latência e melhorar a qualidade do serviço (*Quality of Service* ou QoS) (ALRAWAIS et al., 2017).

Fog procura se beneficiar tanto da proximidade dos dispositivos *edge* aos dispositivos *endpoint* (ou pontos de extremidade) quanto da escalabilidade de recursos sob demanda da nuvem. De acordo com Gupta et al. (2017), dentre os principais benefícios oferecidos por este paradigma podemos destacar:

- Fog permite filtrar e analisar os dados gerados pelos sensores utilizando dispositivos *edge*, reduzindo drasticamente a quantidade de dados enviados à nuvem;
- Caso todos os dados gerados pelos sensores forem enviados diretamente à nuvem, pode surgir um gargalo de desempenho. Como fog permite filtrar e processar uma quantidade significativa de dados dos sensores, a arquitetura de processamento de dados pode ser modelada de forma distribuída e escalável; e,
- O processo de comunicação entre os sensores e a nuvem pode ser lento ou estar indisponível caso ocorram problemas de comunicação. Ao processar os dados localmente na fog podemos obter uma redução considerável na latência, permitindo o processamento de dados em tempo real.

Fog pode ser também ser considerada como uma extensão da nuvem que, ao mesmo tempo, introduz novos desafios de segurança e privacidade e oferece uma plataforma ideal para tratar múltiplas questões de segurança e privacidade presentes em redes IoT. O poder computacional de nós fog pode ser utilizado para prover criptografia

para os dispositivos IoT localizados abaixo desse nó na topologia de rede e, desta forma, oferecer um novo nível de segurança às redes IoT. (ALRAWAIS et al., 2017).

2.4. Introdução à inteligência artificial

Na última década, sistemas de inteligência artificial e aprendizado de máquina obtiveram desempenho excelente em tarefas anteriormente consideradas computacionalmente impossíveis. Foram implementados em áreas como reconhecimento de voz, classificação de imagens e jogos de tabuleiro; mas também em áreas críticas, como finanças, medicina, carros autônomos e recomendações de conteúdo. Estes sistemas devem satisfazer certos critérios, como imparcialidade, confiabilidade, segurança, privacidade e usabilidade (DOSILOVIC; BRCIC; HLUPIC, 2018).

Dosilovic, Brcic e Hlupic (2018) classificam os diversos modelos de inteligência artificial de acordo com a opacidade em três categorias:

- Modelos de IA de baixa complexidade, como modelos lineares, árvores de decisão e regras; estes modelos são comuns no meio acadêmico e comercial.
- Modelos opacos cuja complexidade não permite desvendar facilmente a lógica nas predições; logo estes modelos são considerados caixas pretas. Esta categoria inclui redes neurais artificiais, *boosted trees* e florestas aleatórias, entre outros.
- Modelos híbridos, que combinam os dois modelos anteriores procurando balancear interpretabilidade e previsibilidade.

Como o aprendizado engloba capacidades de inteligência humana, para criar uma IA devemos lidar com problemas de linguagem natural, raciocínio automatizado e aprendizado de máquina. Segundo Luger (2009), podemos classificar os diversos métodos de aprendizado em quatro principais grupos:

- Baseados em símbolos, onde entidades e relacionamentos de um problema são representados por um conjunto de símbolos. Algoritmos procuram inferir generalizações lógicas e válidas que podem ser expressas por meio destes símbolos.
- Abordagens conexionistas, inspiradas pela arquitetura do cérebro humano. Representam conhecimento como padrões de atividade em redes de pequenas unidades de processamento que, ao modificarem a sua estrutura e pesos, aprendem com base em dados de treinamento. Em lugar de utilizar as generalizações oferecidas por uma linguagem simbólica, modelos conexionistas reconhecem padrões nos dados e os refletem na sua própria estrutura.
- Algoritmos genéticos, inspirados pela genética e teoria da evolução de Darwin. Começam com uma população de possíveis soluções que são avaliadas de acordo com a sua habilidade de resolver o problema, logo as melhores soluções têm maior probabilidade de sobreviver e de combinar-se entre si para gerar a próxima geração de soluções possíveis.
- Abordagens estocásticas, que são baseadas na estatística ou, mais especificamente, na regra de Bayes, onde a experiência condiciona as expectativas para interpretar novos dados. Utilizam-se principalmente modelos Markovianos para demonstrar a interpretação de conjuntos em raciocínio que afetam como a resposta de um agente pode ser encorajada ou desencorajada por feedback.

Para este trabalho, a técnica de inteligência artificial escolhida foi a abordagem conexionista. Esta técnica foi selecionada porque, de acordo com Luger (2009), redes neurais são apropriadas para reconhecimento de padrões e não requerem o uso explícito de símbolos por parte do programador, mas aprendem por meio de exemplos. Esta última característica é de grande interesse, pois seria inviável reprogramar o modelo proposto continuamente, de forma manual. Outras técnicas, como as simbólicas e estocásticas, não são de interesse para este trabalho, pois não se conhece, a priori, algum modelo matemático que represente uma solução para este problema, fazendo-se necessário um estudo detalhado sobre o sistema a fim de formular tal modelo.

2.5. Redes Neurais

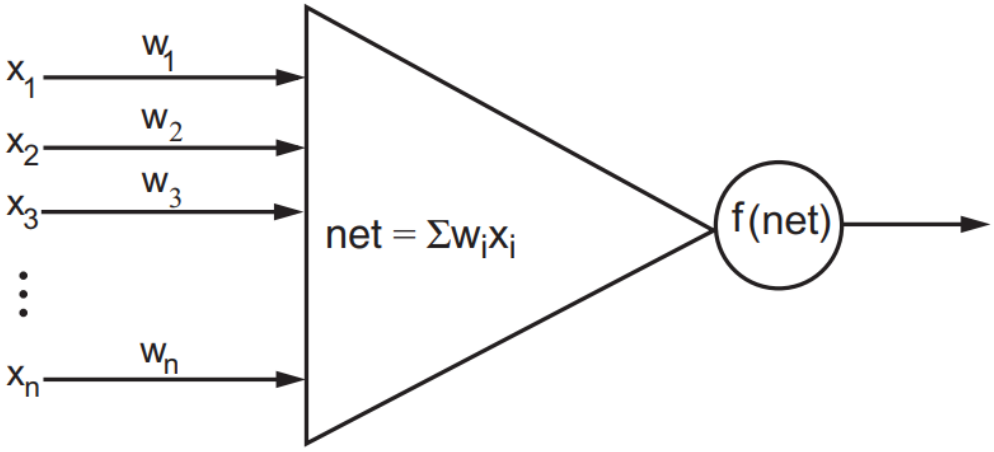
Em particular, nas redes neurais, a escolha de um padrão de codificação pode ser crucial para o aprendizado, já que tanto os padrões quanto as conexões entre componentes são representados por valores numéricos. Uma das principais vantagens desta abordagem é que redes neurais são treinadas (ou condicionadas) em lugar de serem programadas explicitamente, podendo assim capturar detalhes do mundo real sem terem sido explicitamente programadas para reconhecê-los (LUGER, 2009).

Esta abordagem é eficiente para aplicações difíceis de representar por modelos simbólicos e permite representar problemas cujo domínio requer habilidades de percepção humana ou não possui uma sintaxe clara. Também possui degradação gradual, ou seja, a perda de elementos ou sinapses não causa falhas na rede como um todo. De acordo com Luger (2009), as principais tarefas para as quais redes neurais são mais adequadas são:

- classificação de dados;
- reconhecimento de padrões;
- recuperação de memórias, inclusive em memória associativa;
- previsão de causas a partir de efeitos;
- otimizações, inclusive respeitando restrições; e,
- filtragem de ruído.

A base das redes neurais é o neurônio artificial que, de acordo com Luger (2009), possui quatro principais componentes, conforme demonstrado na figura 1:

Figure 1. Componentes principais de um neurônio artificial. (LUGER, 2009)



- sinais de entrada x_i, que podem vir do meio externo ou de outros neurônios. Geralmente possuem valores no conjunto {-1,1} ou são números reais. Alguns destes sinais podem ser constantes (chamados de *bias* ou viés).
- pesos w_i, valores reais que refletem a força de uma conexão.
- nível de ativação $\Sigma w_i\ x_i$, é a soma ponderada das entradas.
- função de ativação f, que pode ser qualquer função contínua e derivável; as mais utilizadas são funções degrau, lineares, sigmodais, logarítmicas ou tangente hiperbólica. Avalia a saída do neurônio comparando o nível de ativação com algum valor limite, e o valor da saída pertence ao conjunto {-1,1} ou aos números reais.

As redes neurais como um todo possuem três características principais:

- A topologia de rede, que representa o padrão de conexões existentes entre os neurônios.
- O algoritmo de aprendizado usado.
- O esquema de codificação e decodificação de valores, utilizados na entrada e saída da rede, respectivamente.

O aprendizado em uma rede perceptron, pode ser supervisionado: a saída da rede neural é comparada com a saída esperada, que é fornecida por um professor (LUGER, 2009). Logo, é calculado o erro entre elas e o neurônio atualiza os pesos para tentar reduzir o erro, segundo a seguinte regra:

$$\Delta w_i = c\left(d - \text{saída}\left(\sum x_i\, w_i\right)\right) x_i$$

Onde:

- c é uma constante que determina a taxa de aprendizado.
- d é o resultado esperado, fornecido pelo professor.
- Δw_i é o ajuste de peso para o i-ésimo componente do vetor de entrada.
- saída $(\sum x_i\ w_i)$ é o valor de saída do perceptron, que é 1 ou -1.

A diferença entre a saída gerada e a esperada será 0, 2 ou -2. Para minimizar o erro médio do conjunto de treinamento, verificamos para cada componente do vetor de entrada:

- se a diferença é nula, não é necessário ajustar os pesos.
- se a saída gerada é -1 e a esperada é 1, incrementamos os pesos na i-ésima linha por $2cx_i$.
- se a saída gerada é 1 e a esperada é -1, decrementamos os pesos na i-ésima linha por $2cx_i$.

Modelos perceptron somente são capazes de resolver problemas linearmente separáveis. Um dos exemplos mais simples de não linearidade é o ou-exclusivo (XOR).

Regra do delta generalizado ou do gradiente descendente: é baseada em uma superfície que representa o erro cumulativo. O eixo y representa o erro acumulado e o eixo x cada possível conjunto de pesos dos neurônios. Para encontrar a direção na qual o erro é reduzido, utilizamos o vetor gradiente, procurando reduzir o erro. Uma desvantagem

deste método é que pode assumir erroneamente que um ponto de mínimo local é ótimo, mesmo não sendo o mínimo global.

Se o valor da taxa de aprendizado for alto, a rede se aproxima mais rapidamente dos pontos ótimos, mas pode não convergir ou oscilar perto dos pontos de mínimo; estes problemas podem ser resolvidos usando um valor menor, porém o aprendizado será mais lento. O valor ótimo varia de acordo com a aplicação, e inclusive pode ser aprimorado usando um fator de momento ou inércia, que pode acelerar a convergência e "pular" alguns mínimos locais.

Modelos perceptron são incapazes de resolver problemas não linearmente separáveis, porém esta limitação pode ser contornada ao empilhar duas ou mais camadas de neurônios (LUGER, 2009). Redes multicamadas são conectadas em camadas, onde os neurônios de uma camada somente passam os seus valores de saída para a próxima camada. Uma rede multicamada possui três tipos de camadas:

- camada de entrada
- camadas intermediárias ou ocultas, constituídas por uma ou múltiplas camadas. O segundo caso é referido como redes neurais profundas ou *deep neural networks* (DNN), em inglês.
- camada de saída

Para ajustar os pesos dos neurônios das camadas intermediárias, o algoritmo de *backpropagation* pode ser usado. Este algoritmo é baseado na regra do delta generalizado e propaga o erro da camada de saída para as camadas intermediárias, avaliando a contribuição feita por cada camada para o erro total na saída. Para múltiplas camadas intermediárias este procedimento pode ser repetido recursivamente. Quando a rede converge lentamente este método pode requerer grande quantidade de poder computacional.

Para realizar o treinamento da rede, o professor separa o conjunto de dados em dois subconjuntos: 70 a 80% dos dados são usados para treinar a rede e os demais para testes e validação.

2.5.1. Keras

De acordo com Chollet et al. (2015), Keras é uma API para a programação de redes neurais em alto nível, que abstrai três linguagens de programação de redes neurais de baixo nível: TensorFlow, CNTK e Theano; sendo TensorFlow a opção selecionada por padrão. É *open-source* e escrita em python.

Oferece suporte para redes neurais rasas, convolucionais e recorrentes; é modular, extensível, flexível e suporta múltiplos *backends*. Permite descrever os modelos por meio da linguagem python, e salvá-los em um arquivo HDF5.

De acordo com Chollet et al. (2015), esta ferramenta é utilizada por grandes empresas, como Netflix e Uber, assim como *startups*. Pode ser implantada em diversas plataformas, como iOS, android, *Java Virtual Machine* e Google Cloud. Possui suporte para paralelismo em GPUs de duas formas: paralelismo de dados e de dispositivos.

Keras oferece diversas opções para tratamento de dados, permitindo normalizar os dados de cada *batch*, e por padrão os dados de treinamento são embaralhados a cada

época. Também oferece otimizadores, como o gradiente descendente estocástico, que suportam momento e taxa de aprendizagem com decaimento.

2.5.2. Tensorflow

De acordo com Abadi et al. (2015), Tensorflow é uma interface, desenvolvida pela Google, para descrever algoritmos de aprendizado de máquina. Esta ferramenta já é usada em pesquisa e em diversas aplicações de aprendizado de máquina, como reconhecimento de voz, visão computacional, processamento de linguagem e classificação de objetos.

Esta ferramenta é flexível e suporta uma variedade de algoritmos de aprendizado de máquina, como algoritmos de treinamento e de inferência para redes neurais. Um programa em Tensorflow pode ser executado em diversos sistemas e dispositivos, com poucas ou nenhuma alteração. Esta característica é valiosa pois o uso de vários sistemas em larga e pequena escala necessitam de um esforço enorme de manutenção.

Tensorflow foi desenvolvido para permitir implementar e testar rapidamente novas ideias e ao mesmo tempo manter alto desempenho. Suporta paralelismo, execução distribuída, operações em GPU e uma grande gama de plataformas heterogêneas de hardware; assim como várias otimizações que aumentam o desempenho e tolerância a faltas.

Tensorflow permite representar redes neurais por meio de um grafo direcionado, descrito em C++ ou Python. Também permite adicionar extensões que oferecem algoritmos de otimização comumente utilizados, como gradiente descendente estocástico e execução parcial de uma rede neural.

3. Trabalhos correlatos

Nesta seção, são apresentados os trabalhos correlatos deste artigo.

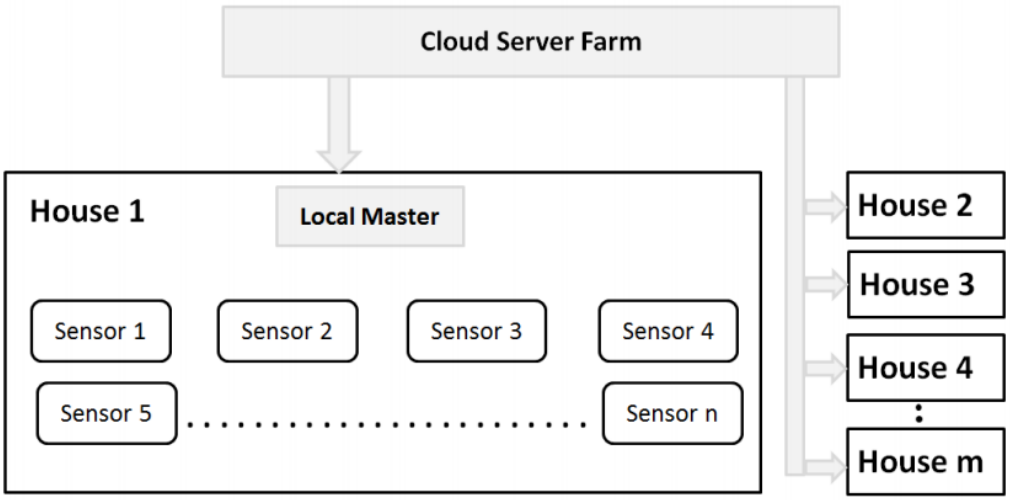
3.1. Power Efficient, Bandwidth Optimized and Fault Tolerant Sensor Management for IOT in Smart Home

Choubey et al. (2015) propoem um framework para tomada de decisões em uma rede IoT aplicada a smart homes. Conforme representado na figura 2, é considerada a utilização de múltiplos nós sensores IoT em diferentes ambientes de uma casa, onde são utilizadas técnicas de IA nos dados coletados para detectar se existem dependências entre os mesmos. Com base nessas dependências, valores de um conjunto de sensores podem ser previstos com base nos demais conjuntos de sensores; e, com estes conhecimentos, os sensores são ajustados em tempo real visando minimizar redundância e consumo de energia; e em casos de falhas de sensores, decisões aproximadas podem ser tomadas com base nos padrões anteriormente observados.

3.2. EiF: Toward an Elastic IoT Fog Framework for AI Services

Fog ainda enfrenta desafios em situações que requerem sensibilidade ao contexto e tomada de decisões automatizadas em tempo real, motivos pelos quais grande parte das plataformas IoT não satisfazem os requisitos da indústria. Com estes desafios em mente, An et al. (2019) propõem o framework EiF (*Elastic Intelligent Fog*). Esta plataforma

Figure 2. Arquitetura proposta. (CHOUBEY et al., 2015)



utiliza redes neurais profundas para filtrar dados desnecessários de forma a tomar decisões em tempo real: os nós fog possuem uma IA que, ao gerenciar sensores e nós inteligentes adjacentes, oferece aos usuários serviços IoT como fluxo inteligente de tráfego. IA também é utilizada para coordenar fog e nuvem procurando prever falhas e degradação de QoS com base nos dados de monitoramento da rede, reconfigurando estes serviços dinamicamente, com o objetivo de melhorar QoS. O processamento de dados ocorre nas *edges*, procurando alcançar respostas em milisegundos. EiF utiliza raciocínio dedutivo e indutivo para extrair informação de texto desestruturado, podendo compreender linguagem natural ao complementar texto com um repositório de conhecimento global que pode ser extraído do contexto.

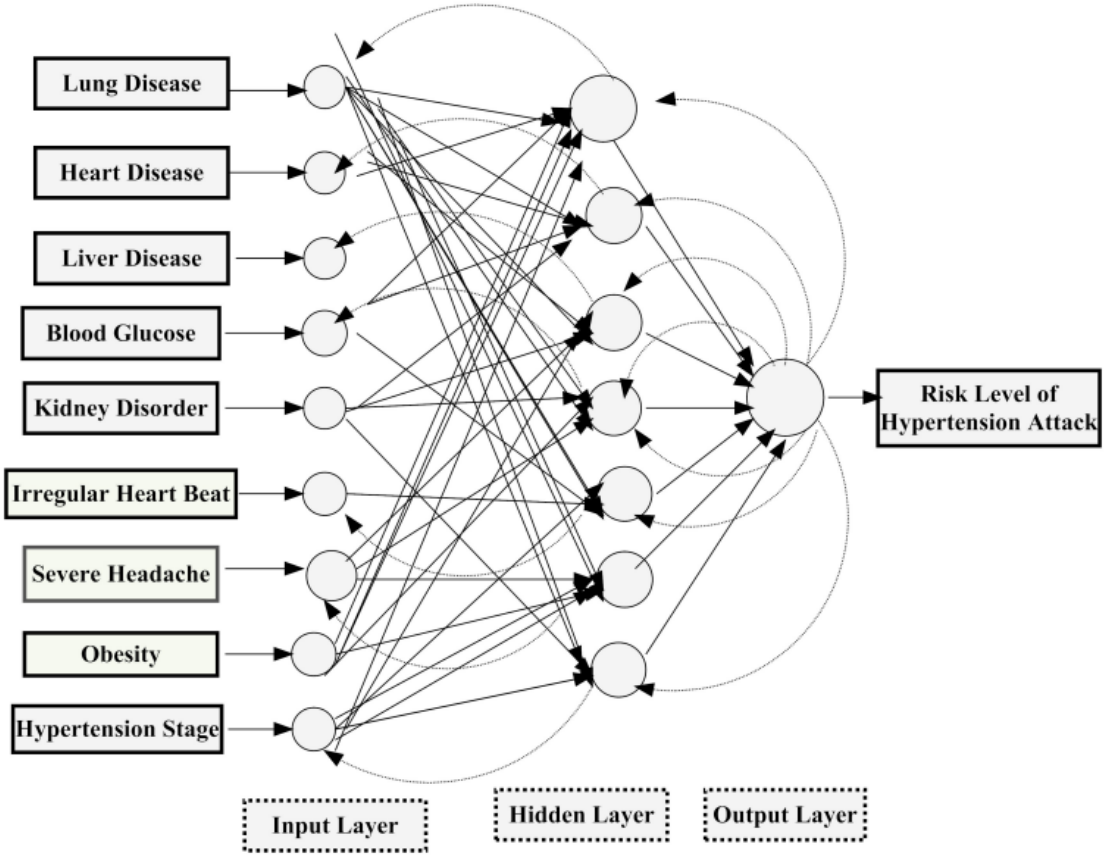
3.3. IoT-Fog based Healthcare Framework to Identify and Control Hypertension Attack

Neste artigo, Sood e Mahajan (2019) propõem um framework IoT-Fog de assistência médica que tem como objetivo identificar crises de hipertensão com base em sintomas do paciente, e notificar emergências aos usuários e médicos em tempo real. Atualmente, para controlar o risco de hipertensão, pacientes devem monitorar certos parâmetros manualmente, visitando centros de saúde; porém este processo pode ser automatizado por meio deste framework no conforto dos seus lares, melhorando a qualidade de vida do paciente. Esta abordagem também pode ser usada para identificar o estágio de hipertensão do paciente por meio de sensores IoT médicos.Este sistema consiste em três subsistemas:

- subsistema de IoT, composto de dispositivos e sensores IoT que capturam os sintomas e os transferem à fog.
- subsistema de fog com *smart gateways*, onde ocorre o processamento e diagnóstico em tempo real. Envia alertas aos telefones celulares dos usuários, possibilitando que medidas de precaução possam ser tomadas a tempo em caso de emergência.
- subsistema de nuvem, onde são armazenados os resultados das análises e históricos médicos dos pacientes, facilitando o acesso aos mesmos por parte de médicos, pacientes e demais usuários do sistema.

Uma rede neural é utilizada na fog para prever o risco do paciente sofrer uma crise de hipertensão. Como representado na figura 3, este risco é calculado com base no estágio de hipertensão e considerando doenças pulmonares, cardíacas, hepáticas e renais, além do nível de glucose no sangue, batimentos cardíacos, dores de cabeça e obesidade. Cada um dos parâmetros anteriores é representado em uma escala entre 0 e 1. Esta rede é treinada por meio de conjuntos de dados onde tanto as entradas quanto as saídas são conhecidas.

Figure 3. Rede neural com *backpropagation* utilizada para prever crises de hipertensão. (SOOD; MAHAJAN, 2019)



3.4. Autonomic IoT Battery Management with Fog Computing

Neste artigo, Sampaio et al. (2019) propõem um sistema de alarme de incêndio para uma casa inteligente, que consiste em um dispositivo IoT e um servidor fog. O dispositivo IoT mede a temperatura e umidade do ar, a concentração de gás, fumaça e presença de chamas; e envia estas condições do ambiente à fog, onde estes dados serão processados, armazenados e exibidos ao usuário por meio de um website. Foi criada uma tabela de estados, onde para cada sensor foi definido um valor de limiar superior que indica a alteração de estado. Foi exemplificado considerando os seguintes valores como limites dos sensores: 35 graus para o sensor de ar; 4% para concentração de gases e presença ou não de chamas. Quando todos os sensores excedem seus respectivas limiares, é considerado um estado de emergência e uma sirene e uma luz LED são acionados. Os dados dos sensores foram capturados de forma independente entre si, por meio de threads em um pipeline, com o objetivo de economizar energia. Durante cada ciclo, o dispositivo

IoT estará acionado durante o tempo de resposta dos sensores, que é de 2 segundos, e logo, se uma emergência não for detectada, entrará no modo de economia de energia até o próximo ciclo. Estimativas de tempo de vida da bateria são apresentadas, tendo em vista o tempo máximo de resposta permitido por lei, que é de 5 segundos. Ao utilizar uma bateria de 2000 mA, este dispositivo tem autonomia estimada de até 18 horas contínuas, e até 45 horas com um ciclo de *sleep* de 3 segundos.

3.5. Design and Implementation of a Power Consumption Management System for Smart Home Over Fog-cloud Computing

Neste artigo, Chen, Azhari e Leu (2018) propõem um modelo de gerenciamento de energia para uma casa inteligente por meio de uma rede fog. A arquitetura proposta consiste na nuvem e três camadas de fog: a primeira camada (camada base) possui vários nodos IoT (tomadas inteligentes), cada nó da segunda camada representa um quarto da casa inteligente, e os nós da terceira camada representam a casa em si. Cada camada de fog é independente das demais, e o seu poder computacional e de armazenamento é proporcional à camada em que se encontra; e a comunicação entre camadas ocorre por meio do protocolo *Zigbee*. As tomadas inteligentes são dispositivos IoT que têm como objetivo medir o consumo energético dos equipamentos conectados à mesma e enviá-los à camada base. A nuvem recebe os dados de uma a quatro vezes por mês, e com eles o usuário pode decidir se o consumo de algum nó deverá ser ajustado de forma a reduzir o consumo de energia da casa. Quando comparado com uma abordagem baseada puramente na nuvem, o framework proposto apresentou tempo de processamento significativamente menor, melhor desempenho e alta eficiência.

4. Desenvolvimento da Proposta

Como proposta, este trabalho considera o contexto de gerenciamento de energia em condomínios inteligentes com o desenvolvimento de um sistema de controle de acesso, assim como o trabalho de gerenciamento de energia de dispositivos IoT do sistema de detecção de incêndio proposto por Sampaio et al. (2019).

A partir dos dados de histórico de acesso, capturados por meio de um sistema de RFID, propomos um sistema inteligente, utilizando a técnica de redes neurais, que tem como objetivo determinar automaticamente os horários em que os apartamentos do condomínio estão desocupados, o que permitiria aumentar o tempo que os dispositivos IoT, de detecção de incêndio, permaneçam em modo de economia de energia e, desta forma, reduzir o consumo (medido em kWh) e o custo (medido em reais). Uma visão geral do modelo proposto é apresentada na figura 4.

A figura 5 ilustra o condomínio considerado, que consiste em 10 edifícios, cada um com 30 apartamentos; onde cada apartamento possui um sensor RFID e um nó de detecção de incêndio, totalizando 600 nós. Os dados são enviados por meio do protocolo *Zigbee*, a um nó fog central para processamento na rede neural.

A figura 6 ilustra o fluxo geral das mensagens no sistema. Os passos indicados são detalhados a seguir:

- 1: O dispositivo IoT detecta a presença de um cartão RFID e lê o seu valor.
- 2: O valor do cartão RFID é encapsulado em um pacote Zigbee e é enviado ao nó fog.

Figure 4. Visão geral do modelo proposto (fonte própria).

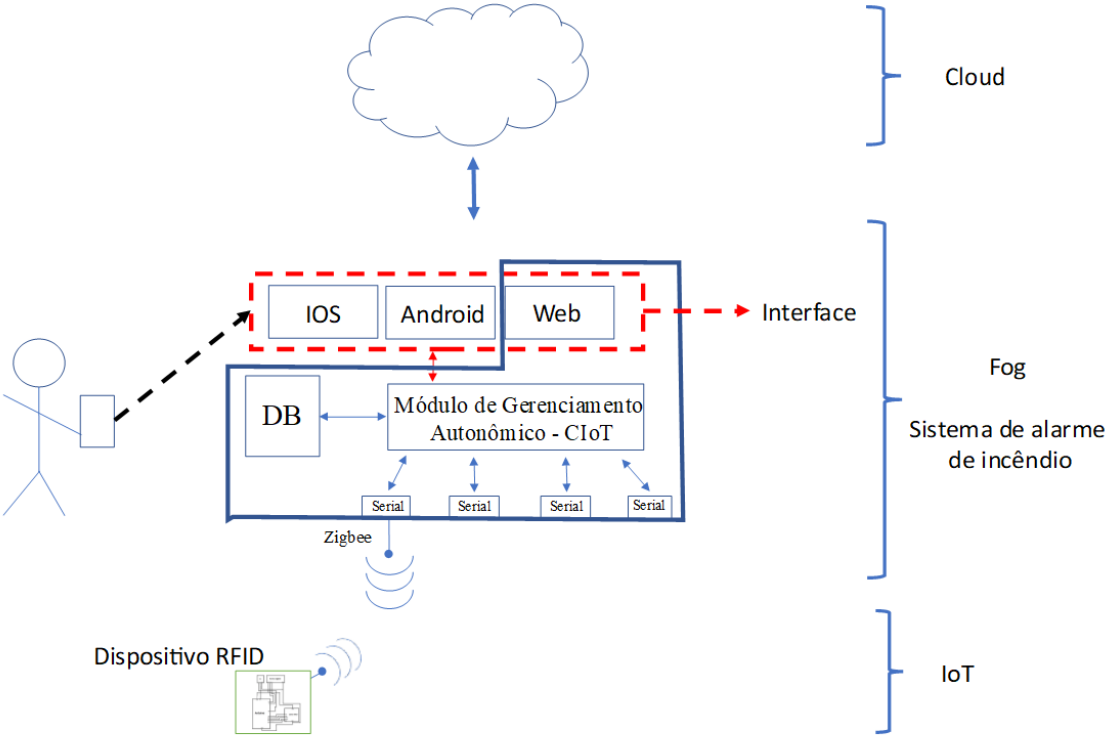
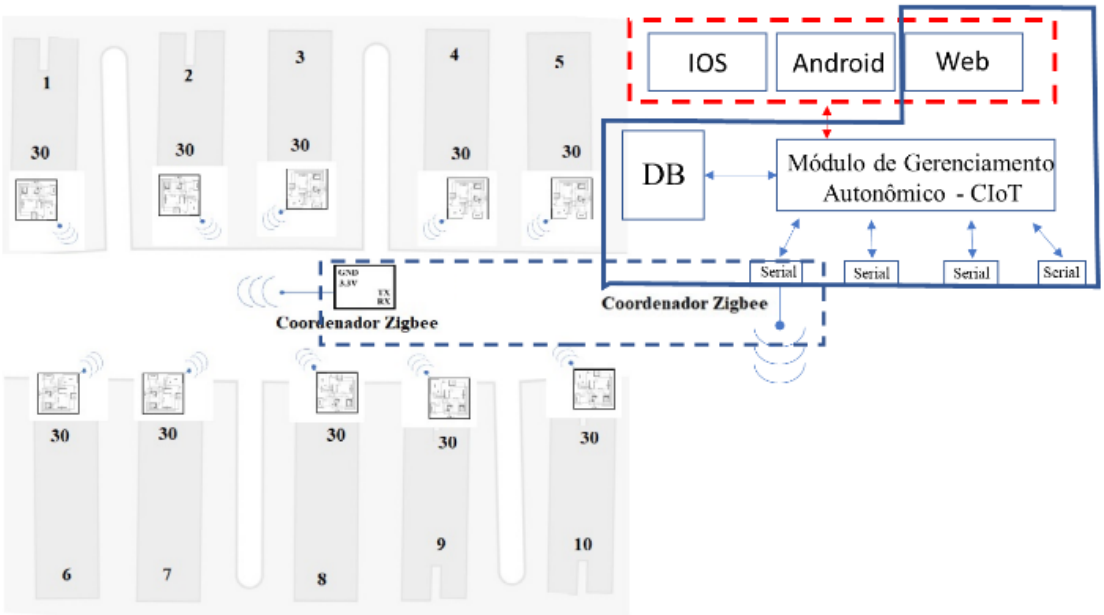
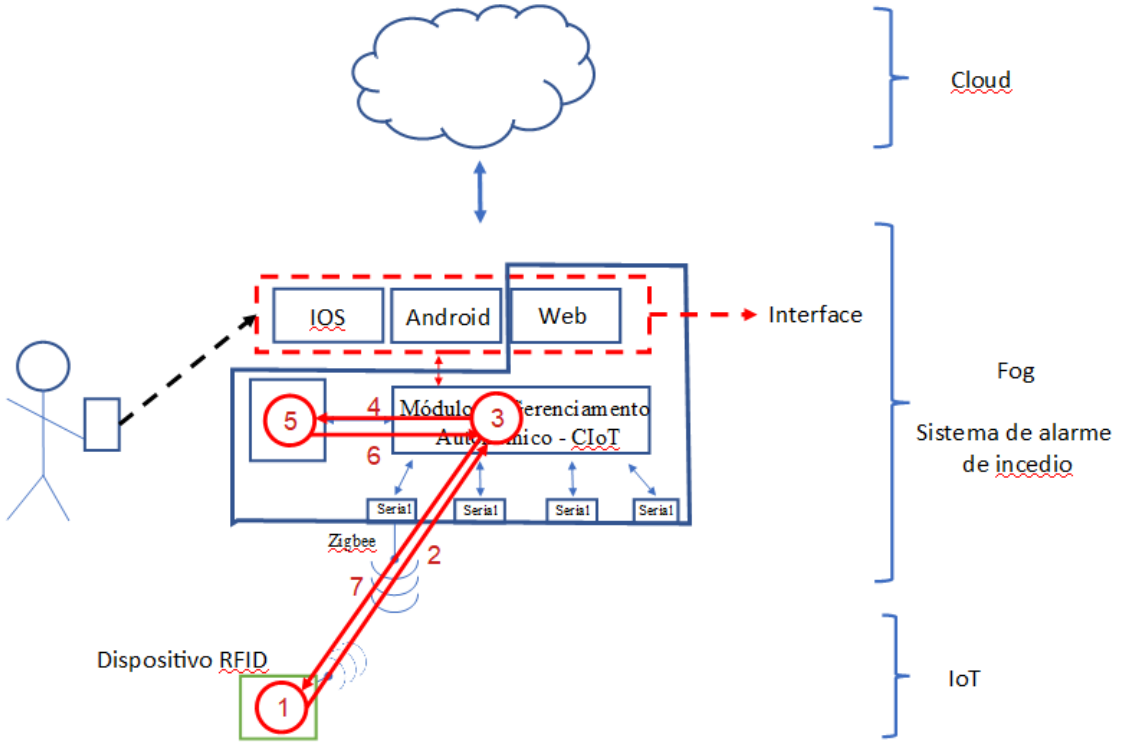


Figure 5. Visão geral do condomínio inteligente considerado para a proposta (fonte própria).



- 3: O nó fog recebe o pacote zigbee, extrai o valor do cartão RFID e o horário em que este pacote foi recebido.
- 4: Estes dois dados são enviados para o banco de dados.
- 5: Os dados são armazenados e é verificado se o cartão é autorizado para acessar

Figure 6. Fluxo geral das mensagens no sistema proposto (fonte própria).



a residência em questão.

- 6: É enviada uma mensagem informando se o cartão possui autorização de acesso ou não.
- 7: Esta informação é repassada para o dispositivo IoT em questão, o qual ativa o relê permitindo o acesso à residência, caso autorizado, ou mostra uma mensagem de erro ao usuário, caso contrário.

Para este trabalho, foi considerado apenas um habitante por apartamento, e os dados iniciais serão obtidos por meio de simulações. Dentre as entradas consideradas pelo sistema podemos mencionar o identificador de cada apartamento, dia e horário. A saída deste sistema será uma variável que indica se a residência estará desocupada nesse momento. O treinamento da rede neural ocorrerá na nuvem e a rede treinada será implementada na rede fog.

4.1. Desenvolvimento do protótipo de nó IoT

É proposto um protótipo do dispositivo IoT com RFID, que é composto por um Arduino Uno, uma antena Zigbee, uma antena RFID e um relê. Este protótipo irá ler o cartão RFID do usuário, registrar o acesso no banco de dados e acionar o relê, liberando a porta para que o usuário possa entrar na sua residência.

Também é proposto um dispositivo fog composto por um Raspberry Pi e uma antena Zigbee, que recebe os pacotes enviados pelo dispositivo IoT e os armazena em um banco de dados, e executa os algoritmos de inteligência artificial.

A rede neural utiliza os dados de histórico de acesso salvos pelo protótipo no banco de dados para inferir os horários em que o usuário estará ou não em sua residência.

Esta informação pode ser usada para ajustar o tempo de *sleep* de outros dispositivos IoT, otimizando o consumo energético.

4.1.1. Desenvolvimento do nó final

Para implementar o dispositivo final, foi utilizada uma placa Arduino Uno, uma antena XBee Zigbee, uma antena RFID MFRC522 e um relê.

4.1.2. Protocolo *Zigbee*

De acordo com Chen, Azhari e Leu (2018), o protocolo *Zigbee* tem como base o padrão IEEE 802.15.4, que define a operação de dispositivos com baixa velocidade de transmissão e baixo consumo energético, quando comparado com outros protocolos utilizados no meio acadêmico e comercial, como Wi-Fi e Bluetooth. De acordo com ZigBee Alliance (2014), dispositivos *Zigbee* podem cumprir os seguintes três papéis:

- Coordenador: cria a rede e configura aspectos como o canal e PAN ID; e é possível utilizar múltiplas redes diferentes em um único local físico. Nesta configuração, os dispositivos não entram em modo de economia de energia (*sleep*).
- Roteador: tem como função distribuir o tráfego na rede. Assim como na configuração coordenador, os dispositivos nesta configuração não entram em modo *sleep*; mas podem armazenar em cache alguns dados referentes aos seus nós filhos, quando estes estiverem em modo *sleep*.
- Dispositivos finais: geralmente possuem sensores, e têm como finalidade capturar os dados lidos por tais sensores e transmiti-los a um nó roteador ou coordenador. Estes dispositivos podem entrar em modo de economia de energia periodicamente.

Para o protótipo de nó IoT, foi escolhida a antena *Xbee Zigbee*, fabricada pela empresa Digi, e suporta o protocolo *Zigbee*. Cada nó permite até 14 nós filhos, tem alcance de 60 metros, frequência de 2.4 GHz e velocidade de transmissão de 250 Kbps.

Para configurar a antena *Xbee*, foi utilizado o aplicativo XCTU, desenvolvido pela Digi. Esta ferramenta permite alterar o papel da antena, PAN ID, endereço de destino dos pacotes, entre outros.

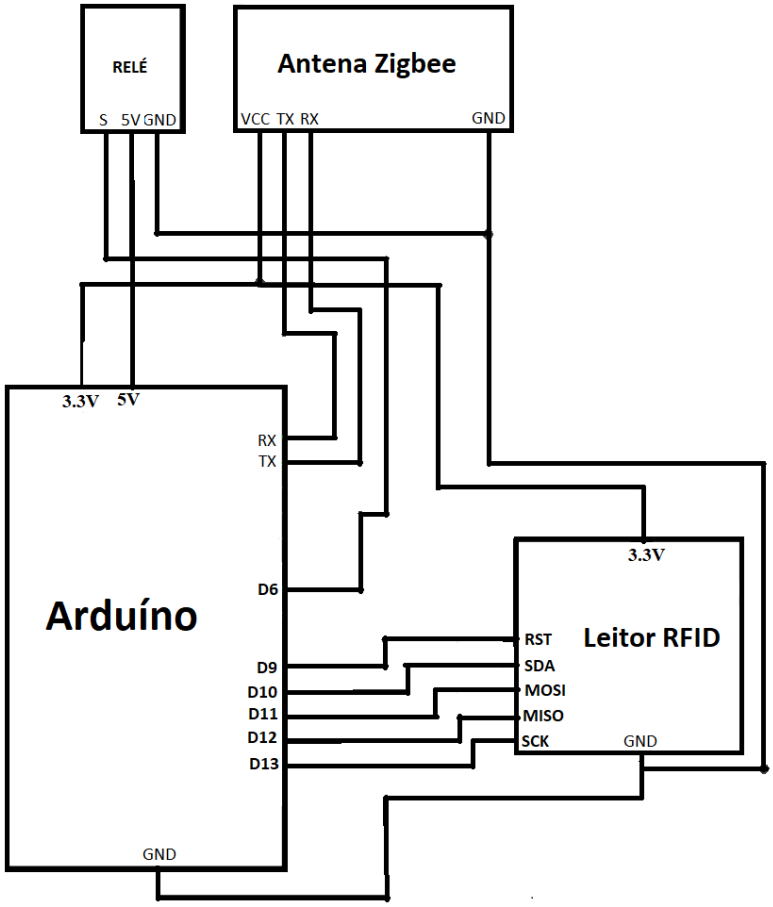
4.2. Desenvolvimento do sistema fog

O nó coordenador possui a antena Xbee e a placa Raspberry Pi. As portas de transmissão (TX) e recebimento (RX) de ambos componentes foram conectadas, além das ligações de energia (3.3 V) e *ground* (GND), conforme demonstrado na figura 7.

A porta serial tanto da antena quanto do Arduino foram configuradas com a taxa de transmissão de 115200 bits/seg. De acordo com Sampaio e Motoyama (2017b), a antena Xbee possui dois modos de funcionamento, configuráveis pelo software XCTU:

- Modo AT (Transparente): este modo é utilizado para enviar dados de um dispositivo final para um coordenador. O coordenador, ao receber estes pacotes pela porta serial RX, os encapsula automaticamente e os envia para o endereço de destino conforme configurado no software XCTU. Para enviar estes dados, basta utilizar o comando *Serial.print()*.

Figure 7. Conexões entre o Arduíno e os demais componentes do dispositivo final (fonte própria).



- Modo API (*Application programming interface*): este modo é utilizado para enviar dados para múltiplos destinatários. É necessário montar os pacotes manualmente, byte a byte, por meio do comando *Serial.write()*.

4.2.1. Montagem de pacotes

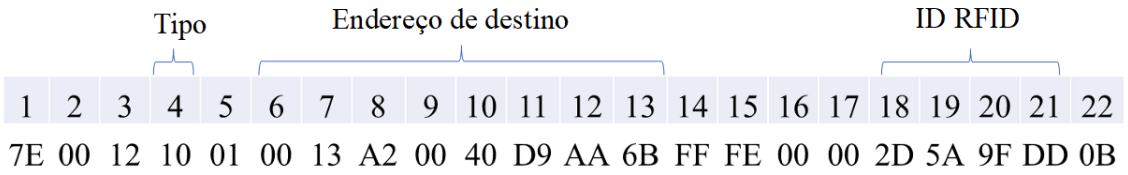
O dispositivo final foi configurado no modo AT e a biblioteca MFRC522, disponibilizada pelo fabricante da antena RFID, foi utilizada para capturar os valores dos tags RFID, conforme ilustrado no algoritmo 1. Este módulo possui um microcontrolador próprio que lê o ID do cartão, representado por 4 bytes, e o envia para o microcontrolador do Arduíno por meio das portas MISO e MOSI. Ao receber este dado, o microcontrolador do Arduíno monta o pacote e o envia pela sua porta TX para a porta RX da antena Zigbee, que então transmite este pacote pela rede (SAMPAIO; MOTOYAMA, 2017a).Os pacotes enviados, chamados de “pacotes de requisição de transmissão”, têm tamanho fixo de 22 bytes neste exemplo.

```
1 for (int i = 0; i < 4; i++) {
2     if (mfrc522.uid.uidByte[i]<16)
      ↪  Serial.print("0");
3     Serial.print(mfrc522.uid.uidByte[i], HEX);
4     }
```

Algoritmo 1: Captura do valor do tag RFID (fonte própria).

Sempre que a antena RFID ler um tag, um pacote será montado e enviado ao Raspberry Pi. Na linha 3 do algoritmo 1, o valor da tag, representado por quatro bytes, é adicionado ao pacote e enviado à antena, que finaliza a montagem do pacote Zigbee adicionando as demais informações que foram pré-configuradas, incluindo o endereço de destino. Ao receber o valor da tag, a antena XBee o transforma em valores hexadecimais, byte a byte; logo estes dados serão encapsulados em um pacote e enviados ao Raspberry Pi. A figura 8 ilustra o pacote criado quando a antena RFID captura o valor *2D 5A 9F DD*.

Figure 8. Exemplo de pacote de requisição de transmissão (fonte própria).



Como o pacote Zigbee foi configurado para ter tamanho fixo, é importante tomar o cuidado de adicionar um zero à esquerda se o valor lido for menor do que 16 (ou 0x10, em hexadecimal), a fim de garantir que dois caracteres sejam sempre enviados por vez.

Ao receber o pacote Zigbee na fog, ele é processado e é verificado se o ID RFID possui permissão de acesso para aquela residência. Em caso positivo, a fog envia um pacote de confirmação para o dispositivo IoT, o qual aciona o relê durante dois segundos. Para este exemplo foi considerado que ao ser acionado o relê, a porta da residência será aberta, permitindo o acesso à residência; e uma vez que tal porta seja fechada, será automaticamente trancada.

A figura 9 descreve detalhadamente a estrutura do pacote da figura 8. É importante notar que os campos de ID do pacote, endereço de destino (tanto de 64 bits quanto de 16 bits), raio de *broadcast* e opções foram configurados anteriormente por meio do software XCTU. Os campos de tamanho do pacote, tipo do pacote e *checksum* são populados automaticamente pelo microcontrolador da antena.

De acordo com Sampaio e Motoyama (2017a), quando a antena XBee é configurada no modo API, os pacotes devem ser montados de forma manual seguindo o padrão do tipo de pacote, já que, ao contrário do modo AT, o encapsulamento não é realizado de forma automática. Esta característica oferece mais liberdade para modificar o pacote.

De acordo com Sampaio e Motoyama (2017b), ao receber o pacote, a antena XBee acoplada ao Raspberry Pi o processa, substituindo o endereço de destino pelo endereço

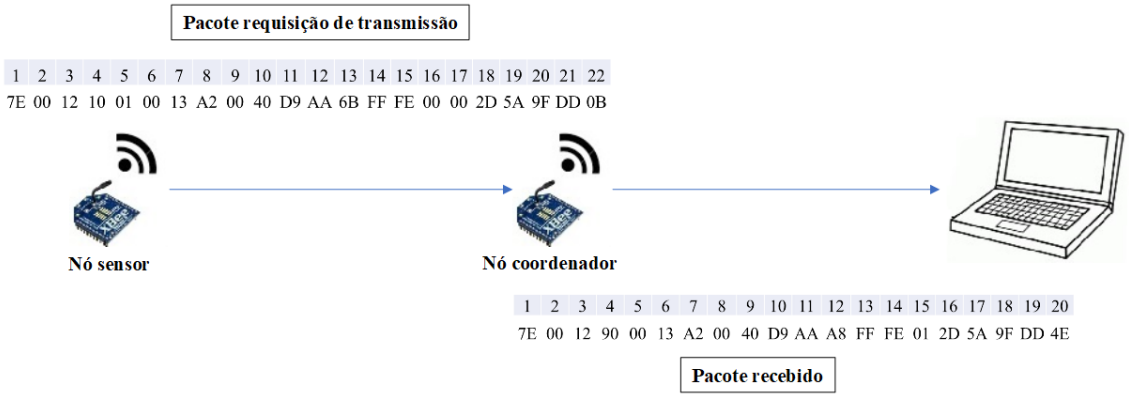
Figure 9. Estrutura do pacote de requisição de transmissão (fonte própria).

Byte	Descrição	Valor hexadecimal	Significado
1	Delimitador inicial	7E	
2 e 3	Tamanho do pacote	00 15	18 bytes de tamanho
4	Tipo do pacote	10	Requisição de transmissão
5	ID do pacote	01	Identificação do pacote
6 a 13	Endereço de destino de 64-bit	00 13 A2 00 40 D9 AA 6B	Endereço do nó coordenador
14 e 15	Endereço de destino de 16-bit	FF FE	Envio para todos roteadores
16	Raio de broadcast	00	Pulos máximos de broadcast
17	Opções	00	
18 a 21	Dados	2D 5A 9F DD	Valor ID cartão RFID
22	Checksum	0B	

do remetente e excluindo os campos de raio de *broadcast* e ID do frame. Logo, o pacote modificado é enviado pela porta RX à porta TX do Raspberry Pi.

A figura 10 representa a transmissão do pacote da figura 8. Assim que a antena XBee destino recebe este pacote, ela o processa e o envia pela porta serial, permitindo obter posteriormente os dados capturados pela antena RFID.

Figure 10. Transmissão e processamento do pacote da figura 8 (fonte própria).



A figura 11 ilustra detalhadamente a estrutura do pacote processado, denominado como pacote de recebimento.

Para este trabalho, a antena do Arduino foi configurada no modo AT, e a do Raspberry Pi no modo API.

Gerenciamento autonômico com IA

O dispositivo IoT proposto, ao ler um tag RFID, envia o valor deste tag para o nó fog. Este valor é recebido por um servidor, escrito na linguagem de programação Python, que envia este dado e o horário de recebimento para um banco de dados MySQL. Neste capítulo é descrito o modelo de armazenamento de dados, a simulação e tratamento de dados, e o desenvolvimento da rede neural.

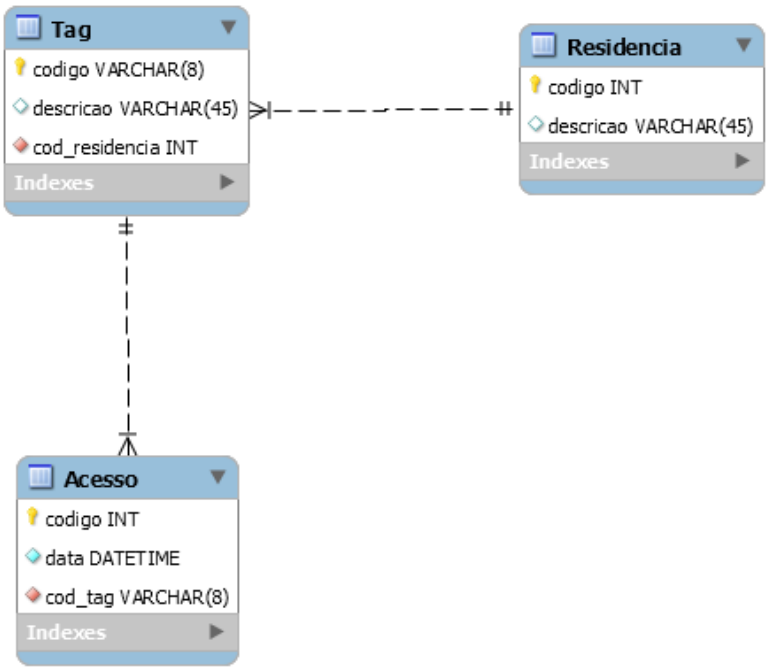
Figure 11. Estrutura do pacote de recebimento (fonte própria).

Byte	Descrição	Valor hexadecimal	Significado
1	Delimitador inicial	7E	
2 e 3	Tamanho do pacote	00 15	21 bytes de tamanho
4	Tipo do pacote	90	Pacote recebido
5 a 12	Endereço de origem de 64-bit	00 13 A2 00 40 D9 AA A8	Endereço do nó IoT
13 e 14	Endereço de origem de 16-bit	FF FE	Endereço desconhecido
15	Opções	00	
16 a 19	Dados	2D 5A 9F DD	Valor ID cartão RFID
20	Checksum	B1	

4.3. Armazenamento de dados

Para armazenar os dados gerados pelo dispositivo IoT, foi escolhido o banco de dados relacional MySQL. Este banco de dados possui três tabelas, conforme ilustrado na figura 12.

Figure 12. Esquema do banco de dados MySQL (fonte própria).



- Residência: representa cada uma das residências do condomínio. Possui um código serial, isto é, um inteiro com incremento automático que identifica unicamente cada residência; assim como um campo para armazenar uma breve descrição da residência.
- Tag: representa cada uma das tags RFID do condomínio. A sua chave primária é o próprio código do tag, que é lido pela antena RFID do dispositivo final. Também possui um campo para adicionar uma breve descrição da tag ou do dono da mesma,

e o identificador único da residência à qual este tag pertence; portanto, cada tag somente pode pertencer a uma residência e possuir um único dono.
- Acesso: esta tabela armazena os acessos às residências. A sua chave primária é um inteiro serial, o atributo *data* armazena a data e hora em que o acesso ocorreu, e o atributo *cod_tag* é o código da tag RFID referente ao acesso.

A priori, devem ser cadastradas residências e tags no banco de dados por meio do terminal MySQL ou por outra ferramenta de gerência de banco de dados. Os acessos são armazenados automaticamente pelo servidor python. É interessante notar que a data e hora são atribuídos pelo servidor e não pelo dispositivo final, reduzindo desta forma o tamanho dos pacotes e a largura de banda requerida.

4.4. Simulação de dados

A fim de treinar a rede neural, foram gerados dados por meio da ferramenta *Libreoffice Calc*. Esta ferramenta permite gerar dados que seguem uma distribuição estatística, como beta, binomial, qui-quadrado, exponencial, gamma, hipergeométrica, lognormal, normal, uniforme e Weibull, entre outras.

Esta ferramenta foi escolhida por permitir a manipulação de dados de forma simples e intuitiva, permitindo utilizar formatos como *datetime*, compatível com SQL; assim como por oferecer suporte a várias funções lógico-matemáticas, como somatório, piso e módulo.

Conforme ilustrado na tabela 1, foram considerados cinco cenários que representam os perfis de grupos de moradores, conforme os horários de entradas e saídas dos mesmos.

Table 1. Cenários simulados, com os seus respectivos cronogramas e quantidade de residências (fonte própria).

Cenário	1	2	3	4	5
Quantidade de residências	150	40	30	60	20
Primeiro acesso (saída)	08:00:00	06:00:00	11:00:00	7:00:00	17:00:00
Segundo acesso (entrada)	11:30:00	13:00:00	16:00:00	9:30:00	02:00:00
Terceiro acesso (saída)	12:30:00		20:00:00	13:00:00	
Quarto acesso (entrada)	18:00:00		22:00:00	15:40:00	
Quinto acesso (saída)				18:00:00	
Sexto acesso (entrada)				20:50:00	

Para cada cenário foram simulados valores de entradas e saídas diários, considerando apenas dias úteis semanais. Dessa forma, para representar um mês foram gerados dados para 22 dias uteis.

Para gerar os valores dos horários de acesso, considera-se uma distribuição normal com a média sendo o valor do acesso representado na tabela 1 e com desvio padrão de 15 minutos.

Assume-se que antes do primeiro acesso, a pessoa está em sua residência, logo o primeiro acesso corresponde a uma saída; o segundo acesso representa o morador retornando à sua residência; o terceiro uma nova saída, e assim por diante.

Ao seguir a configuração de números no padrão brasileiro, a separação decimal corresponde a uma vírgula. Esta característica ocasionou dois problemas durante o desenvolvimento das simulações:

- Como o formato CSV também utiliza a vírgula por padrão para separar campos da tabela, a estrutura da tabela era corrompida. Para resolver este problema, o separador de campos do arquivo CSV foi alterado para um ponto e vírgula (;).
- Posteriormente, ao importar os dados salvos neste padrão para a rede neural, estes números eram erroneamente identificados como *string*, pois Python segue o formato americano, onde a separação decimal é simbolizada por um ponto. Para tratar este problema, a configuração foi alterada para o padrão americano nos campos que serão utilizados pela rede neural.

Também é importante destacar que foram encontrados problemas causados por erros de arredondamento numéricos, portanto, funções de comparação foram reformuladas de forma a utilizar operadores de comparação, como maior e menor, em lugar do operador de igualdade.

Após criar a simulação, estes dados foram exportados no formato CSV. Este formato armazena os dados de uma tabela em um arquivo de texto, separando-os por vírgula, permitindo a manipulação dos mesmos por outro programa, de forma simples.

4.4.1. Tratamento de dados

As entradas e saídas da rede neural devem ser representadas em formato numérico. De acordo com Kuźniar e Zajac (2015), o método de preprocessamento de dados é um fator essencial para determinar o sucesso de uma aplicação de redes neurais. Com este objetivo, após executar a simulação, os dados foram codificados no formato numérico *float*, no software *Libreoffice Calc*, e logo armazenados em formato CSV, para a sua posterior aplicação na rede neural. A codificação utilizada para as variáveis é a seguinte:

- A data foi representada como a quantidade de dias decorridos desde primeiro de janeiro de 1900.
- A hora, incluindo minutos e segundos, foi codificada como um número real, pertencente ao conjunto [0, 1).
- O tag foi representado como um número decimal positivo.
- Para representar se a residência está desocupada ou não, foi utilizada uma variável que assume o valor 1 quando desocupada e 0 quando não.

4.5. rede neural

Para implementar a rede neural, foi utilizado o API Keras, que pode ser considerado como um *wrapper* para a linguagem de programação de redes neurais de baixo nível TensorFlow. A rede em si foi descrita na linguagem Python. Neste capítulo é apresentada uma breve introdução ao Keras e ao Tensorflow, assim como uma descrição detalhada da implementação da rede neural.

4.5.1. Desenvolvimento da rede neural

O algoritmo 2 descreve a rede neural. É importante destacar que as funções utilizadas são implementadas pelo Keras, que por sua vez utiliza o Tensorflow. Este algoritmo é detalhado, passo a passo, a seguir:

```
1  dataset =
    ↪  genfromtxt(r'../simulacao/csv/cenarios_1_a_6.csv',
    ↪  encoding='latin-1', delimiter=',',
    ↪  skip_header=2,
2                          usecols=(2, 3, 4, 5))
3  X = dataset[:, 0:3]
4  Y = dataset[:, 3]
5
6  model = Sequential()
7  model.add(Dense(12, input_dim=3, activation='relu'))
8  model.add(Dense(24, activation='relu'))
9  model.add(Dense(1, activation='sigmoid'))
10
11 model.compile(loss='mean_squared_error',
    ↪  optimizer='sgd', metrics=['accuracy'])
12
13 model.fit(X, Y, epochs=30, batch_size=10)
14 _, accuracy = model.evaluate(X, Y)
```

Algoritmo 2: Código da rede neural em Python com Keras (fonte própria).

- A linha 1 representa a importação dos dados do arquivo CSV. Conforme detalhado na seção 4.4.1, os dados estão armazenados neste arquivo em formato numérico.
- A linha 3 separa as primeiras três colunas (0, 1 e 2) do arquivo CSV para servirem como entrada da rede neural. Estas colunas representam o dia, hora e tag.
- A linha 4 separa a coluna 3 para servir como a saída da rede neural. Este dado corresponde a uma variável que indica se a residência está desocupada no momento, conforme descrito na seção 4.4.1.
- A linha 6 indica que o modelo da rede neural é sequencial.
- A linha 7 descreve a camada de entrada. O primeiro parâmetro é a quantidade de neurônios desta camada, o segundo indica que a rede possui três entradas, e o último indica o modo de ativação da rede, neste caso, unidade linear retificada.
- A linha 8 representa a camada intermediária ou oculta, com 24 neurônios e como modo de ativação a unidade linear retificada. Como esta rede é rasa, ela possui uma única camada intermediária.
- A linha 9 representa a camada de saída. Possui um único neurônio, que corresponde à saída da rede e o modo de ativação é sigmoidal.
- A linha 11 indica como o modelo será compilado. O primeiro argumento é a função de perda, neste caso, erro quadrático médio. O segundo argumento indica o otimizador, neste caso gradiente descendente estocástico. O último parâmetro

representa a função de métrica, isto é, a função que mede o desempenho da rede; neste caso, foi escolhida a acurácia.

- A linha 13 indica como o modelo será treinado. O primeiro argumento corresponde aos dados de entrada, o segundo aos de saída, o terceiro indica a quantidade de épocas e o último indica o tamanho de lote, ou *batch*.
- A linha 14 avalia o treinamento da rede com respeito à acurácia.

A rede neural realiza a tarefa de classificação, pois dado um dia, horário e tag, a rede irá determinar se a residência está desocupada (valor 1) ou ocupada (valor 0).

Com o objetivo de encontrar uma configuração que apresentasse uma melhor acurácia, foram testadas diversas variações dos seguintes parâmetros: número de camadas, quantidade de neurônios de cada camada, modos de ativação, funções de perda e otimizadores. Alguns desdes testes e as suas respectivas acurácias são apresentados na tabela 2.

Os parâmetros da solução escolhida correspondem aos apresentados na primeira linha da tabela 2, assim como na figura 13. Nesta figura, a entrada é composta por três dados, e o indicador *units* representa a quantidade de neurônios da camada. Após o treinamento da rede neural com os dados descritos na seção 4.4, a precisão obtida da rede foi de 65,65%.

A figura 14 apresenta a acurácia da rede neural escolhida em relação à quantidade de épocas. Como podemos ver, a acurácia se mantém estável após a quinta época, logo uma quantidade maior de épocas é desnecessária. A figura 15 apresenta a taxa de perda representada por meio do erro quadrático médio. Como podemos ver, o erro fica aproximadamente estável a partir da quinta época, reforçando a conclusão obtida pela figura 14. Com estas informações, a quantidade de épocas escolhida para treinar a rede neural foi de 10 épocas.

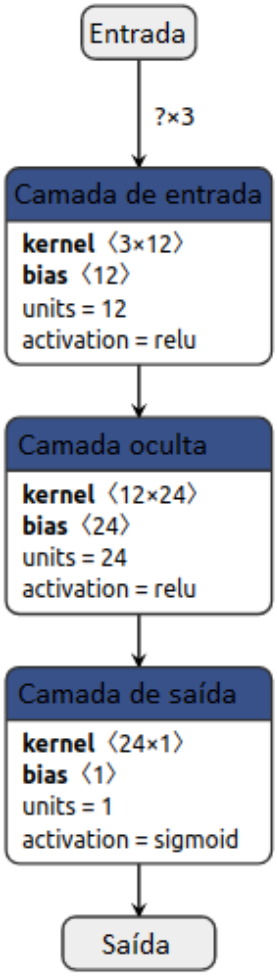
Table 2. Alguns modelos de redes neurais com as suas respectivas acurácias (fonte própria).

Camada 1	Camada 2	Camada 3	Camada 4	Acurácia (%)
12	24	1		65,65
12	60	1		34,35
12	12	1		34,35
12	5	1		34,35
12	24	24	1	65,65
12	60	60	1	34,35
12	5	5	1	34,35

5. CONCLUSÕES E TRABALHOS FUTUROS

Para este trabalho, foi utilizada uma rede neural com três camadas, e a precisão obtida foi de 65,65%. Esta precisão poderia ser melhorada ao realizar uma análise mais detalhada de redes neurais, utilizar mais dados para o treinamento, e possívelmente ao utilizar redes neurais profundas ou convolucionais.

Figure 13. Modelo da rede neural desenvolvida (fonte própria).



Uma otimização adotada com o objetivo de economizar banda é que a data e hora são definidas pelo servidor, e não pelo dispositivo IoT. Desta forma, o único dado enviado pela rede é o tag RFID.

O conhecimento gerado pela rede pode ser utilizado para diversas aplicações, possibilitando aumentar a qualidade de vida dos moradores e diminuir desperdícios; assim como possibilitar o gerenciamento autônomo de energia e outros recursos.

Dentre os trabalhos futuros, podemos citar:

- Utilizar redes neurais profundas ou convolucionais e procurar obter melhor acurácia na predição de dados.
- Além dos parâmetros utilizados neste trabalho, considerar também outras variáveis que possam alterar a rotina dos moradores, como alguns sensores meteorológicos, de entrada para a rede neural. Também podem ser consideradas outras condições, como feriados, dias letivos, finais de semana, etc.
- Com o fim de obter dados mais relevantes para o desenvolvimento do trabalho, utilizar dados reais, ou simulações mais sofisticadas, para o treinamento da rede

Figure 14. Acurácia obtida pela rede neural em relação à quantidade de épocas (fonte própria).

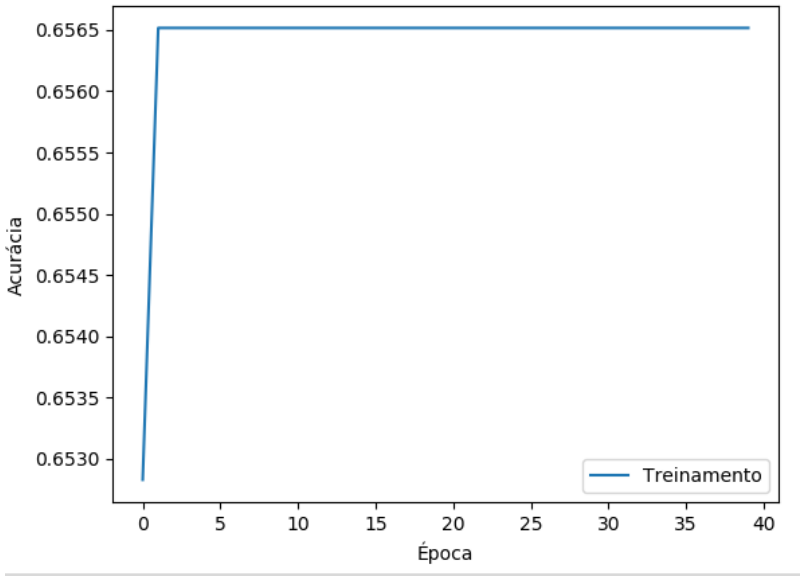
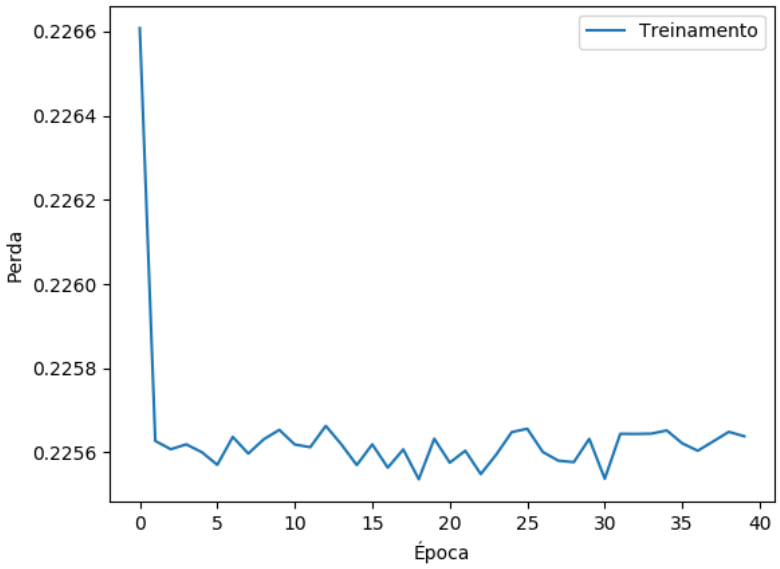


Figure 15. Perda da rede neural em relação à quantidade de épocas (fonte própria).



neural.

- Integrar completamente o banco de dados à rede neural.
- Desenvolver uma interface web para oferecer os demais serviços no banco de dados, como cadastrar residências, tags, habitantes, e outros.
- Expandir este trabalho para suportar múltiplos habitantes e múltiplos tags por residência.
- Utilizar as informações obtidas pela rede neural para realizar o gerenciamento autônomo da residência.
- Utilizar estes conhecimentos para realizar o gerenciamento autônomo de energia,

procurando reduzir desperdícios.

- Utilizar estes conhecimentos para otimizar a quantidade de dados gerados e armazenados no banco de dados.
- Estes conhecimentos também podem ser utilizados para uma infinidade de aplicações que melhorem a qualidade de vida das pessoas. Por exemplo, acionar o sistema de ar condicionado em um dia quente, alguns minutos antes da chegada de um morador, de forma a oferecer uma temperatura ambiente agradável imediatamente após a chegada do mesmo.

References

ABADI, M. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. Software available from tensorflow.org. Disponível em: ⟨https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45166.pdf⟩.

ALRAWAIS, A. et al. Fog computing for the internet of things: Security and privacy issues. *IEEE Internet Computing*, v. 21, p. 34–42, 2017.

AN, J. et al. Eif: Toward an elastic iot fog framework for ai services. *IEEE Communications Magazine*, v. 57, n. 5, p. 28–33, 05 2019. Disponível em: ⟨https://www.researchgate.net/publication/333069163⟩.

CHEN, Y. D.; AZHARI, M. Z.; LEU, J. S. Design and implementation of a power consumption management system for smart home over fog-cloud computing. *IGBSG 2018 - 2018 International Conference on Intelligent Green Building and Smart Grid*, IEEE, p. 1–5, 2018.

CHOLLET, F. et al. *Keras documentation*. 2015. Disponível em: ⟨https://keras.io/⟩.

CHOUBEY, P. K. et al. Power efficient, bandwidth optimized and fault tolerant sensor management for iot in smart home. *Souvenir of the 2015 IEEE International Advance Computing Conference, IACC 2015*, p. 366–370, 2015. Disponível em: ⟨https://www.researchgate.net/publication/283125637⟩.

DINH, H. T. et al. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless Communications and Mobile Computing*, v. 13, n. 18, p. 1587–1611, 2013. Disponível em: ⟨https://onlinelibrary.wiley.com/doi/abs/10.1002/wcm.1203⟩.

DOSILOVIC, F. K.; BRCIC, M.; HLUPIC, N. Explainable artificial intelligence: A survey. *2018 41st International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2018 - Proceedings*, p. 210–215, 2018. Disponível em: ⟨https://www.researchgate.net/publication/325398586⟩.

GUPTA, H. et al. iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments. *Software - Practice and Experience*, v. 47, n. 9, p. 1275–1296, 2017. ISSN 1097024X.

IORGA, M.; MARTIN, M. J.; FELDMAN, L. Fog computing conceptual model nist special publication 500-325. p. 11, 2018. Disponível em: ⟨https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-325.pdf⟩.

JENNINGS, B.; STADLER, R. Resource Management in Clouds: Survey and Research Challenges. *Journal of Network and Systems Management*, v. 23, n. 3, p. 567–619, 2015. ISSN 10647570. Disponível em: ⟨https://doi.org/10.1007/s10922-014-9307-7⟩.

KUźNIAR, K.; ZAJaC, M. Some methods of pre-processing input data for neural networks. *Computer Assisted Methods in Engineering and Science*, v. 22, p. 141–151, 2015.

LUGER, G. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. University of New Mexico, New Mexico, United States: Pearson Education Inc, 2009. v. 9. 754 p. ISSN 1895-1767. ISBN 9780321545893.

RAY, P. P. A survey on Internet of Things architectures. *Journal of King Saud University - Computer and Information Sciences*, King Saud University, v. 30, n. 3, p. 291–319, 2018. ISSN 22131248. Disponível em: ⟨https://doi.org/10.1016/j.jksuci.2016.10.003⟩.

SAMPAIO, H.; MOTOYAMA, S. Implementation of a greenhouse monitoring system using hierarchical wireless sensor network. *2017 IEEE 9th Latin-American Conference on Communications, LATINCOM 2017*, v. 2017-January, p. 1–6, 2017.

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, p. 89–103, 2019. ISSN 16113349.

SOOD, S. K.; MAHAJAN, I. Iot-fog-based healthcare framework to identify and control hypertension attack. *IEEE Internet of Things Journal*, v. 6, n. 2, p. 1920–1927, April 2019. ISSN 2327-4662.

YASUMOTO, K.; YAMAGUCHI, H.; SHIGENO, H. Survey of Real-time Processing Technologies of IoT Data Streams. *Journal of Information Processing*, v. 24, n. 2, p. 195–202, 2016.

ZigBee Alliance. *ZigBee-PRO Stack Profile: Platform restrictions for compliant platform testing and interoperability*. 2014. Disponível em: ⟨https://zigbee.org/download/standard-zigbee-pro-specification/⟩.

6. Apêndice: Código

6.1. Banco de dados

6.1.1. banco_de_dados.sql

```
create user '123' identified by '1234';
grant all on *.* to '123';

create database 'lrg';

create table residencia(codigo serial, descricao
 ↪  varchar(45));

create table tag(codigo varchar(8), descricao
 ↪  varchar(45), cod_residencia bigint unsigned,
 ↪  primary key (codigo), foreign key
 ↪  (cod_residencia) references
 ↪  residencia(`codigo`));

create table acesso(codigo serial, data datetime,
 ↪  cod_tag varchar(8) unique not null, foreign key
 ↪  (cod_tag) references tag(codigo));
```

6.2. Rede neural

6.2.1. rede_neural.py

```
from numpy import genfromtxt
import numpy
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dense
from keras.preprocessing.text import
 ↪  text_to_word_sequence
from keras.preprocessing.text import one_hot

if __name__ == '__main__':

    dataset =
     ↪  genfromtxt(r'../simulacao/csv/cenarios_1_a_6.csv',
     ↪  encoding='latin-1', delimiter=',',
     ↪  skip_header=2,
                          usecols=(2, 3, 4, 5))
```

```
    X = dataset[:, 0:3]  # ultima linha - skip_header
    Y = dataset[:, 3]

    for i in range(0, len(X)):
        for j in range(0, len(X[0])):
            if numpy.isnan(X[i][j]):
                print(i, j)

    model = Sequential()
    model.add(Dense(12, input_dim=3, activation='sigmoid'))
    model.add(Dense(24, activation='sigmoid'))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(loss='mean_squared_error', optimizer='sgd', metrics=['accuracy'])

    history = model.fit(X, Y, epochs=10, batch_size=10)

    _, accuracy = model.evaluate(X, Y)
    print('Accuracy: %.2f' % (accuracy * 100))

    predictions = model.predict_classes(X)

    # summarize some cases
    #for i in range(60):
    #    print('%s => %d (expected %d)' % (X[i].tolist(), predictions[i], Y[i]))

    # model.save('modelo.H5')

    '''
    # graficos de acuracia e validacao
    plt.plot(history.history['acc'])
    plt.ylabel('Acurácia')
    plt.xlabel('Época')
    plt.legend(['Treinamento'])
    plt.show()

    plt.plot(history.history['loss'])
    plt.ylabel('Perda')
    plt.xlabel('Época')
    plt.legend(['Treinamento'])
    plt.show()
    '''
```

6.3. Leitor serial

6.3.1. main.py

```
import serial
import pymysql
import time
import socket
from threading import Thread, Lock
from conf import *
from pdu import *
import datetime

DEVICE_LAST_TIME =
 ↪  dict.fromkeys(DEVICE_INTERVAL.keys(), 0)
BLACKLIST = set()


def serial_read(port, mutex=Lock()):

    while SERIAL_READ:
        table_name = TB_NAME
        attributes = []
        values = []

        pdu = PDU_DEFAULT['attribute']
        size = PDU_DEFAULT['size']
        payload = PDU_DEFAULT['payload']

        address = ""

        insert = True
        i = 0
        mutex.acquire()
        while i < len(pdu):

            read = port.read(size[i])

            if pdu[i] == 'pdu':
                try:
                    pdu_type = PDU_TYPE[read]
                except:
                    insert = False
                    break

                size = pdu_type['size']
                pdu = pdu_type['attribute']
                payload = pdu_type['payload']
                i = 0
                continue
```

```
            if pdu[i] in payload:
                if pdu[i] == 'endereco':
                    read = read.hex().upper()
                    address = read
                else:
                    read = read.decode('utf-8')

                attributes.append('data')

                 ↪  values.append(datetime.datetime.today())
                attributes.append(pdu[i])  # tag
                values.append(read)
            i += 1
        mutex.release()
        if insert:
            insert_database(table_name, attributes,
             ↪  values)
            print(values)


def insert_database(table_name, attributes, values):
    placeholders = ', '.join(['%s'] *
     ↪  len(attributes))
    attributes = ', '.join(attributes)
    values = tuple(values)
    print(values)

    connection = pymysql.connect(
        host=DB_HOST, user=DB_USER,
         ↪  password=DB_PASSWORD, db=DB_NAME
    )
    try:
        with connection.cursor() as cursor:
            query = " INSERT INTO %s ( %s ) VALUES (
             ↪  %s ) " % (
                table_name, attributes, placeholders
            )
            print(query, table_name, attributes,
             ↪  placeholders)
            cursor.execute(query, values)
        connection.commit()

    finally:
        connection.close()
```

```
def device_control(address):
    try:
        time_in = DEVICE_LAST_TIME[address]
        time_fn = time.time()
        interval = time_fn - time_in
        if interval < DEVICE_INTERVAL[address]:
            BLACKLIST.add(address)
        DEVICE_LAST_TIME[address] = time.time()
    except:
        BLACKLIST.add(address)

    return address not in BLACKLIST


def server(port, mutex):
    server_socket = socket.socket()
    server_socket.bind((SERVER_HOST, SERVER_PORT))
    server_socket.listen()
    while True:
        conn = server_socket.accept()[0]
        data = conn.recv(4096).decode('utf-8')
        if 'A' in data:
            mutex.acquire()
            port.write(b'1')
            mutex.release()
        elif 'D' in data:
            mutex.acquire()
            port.write(b'0')
            mutex.release()

        conn.close()
    server_socket.close()


if __name__ == "__main__":
    port = serial.Serial(SERIAL_PORT,
     ↪  SERIAL_BAUDRATE)
    mutex = Lock()

    serial_thread = Thread(target=serial_read,
     ↪  args=(port, mutex))
    serial_thread.start()
    #server_thread = Thread(target=server,
     ↪  args=(port, mutex))
    #server_thread.start()

    serial_thread.join()
    #server_thread.join()
```

6.3.2. pdu.py

```
# --------------------------------
#     PDU DESCRIPTION
# --------------------------------
# forma de organizacao do pacote, criar um novo
PDU_DEFAULT = {
    'attribute': ['pdu'],
    'size': [1],
    'payload': [],
    'length': 1
}

PDU_ANA = {
    'attribute': ['endereco', 'umidade',
    ↪  'temperatura', 'gas', 'chamas'],
    'size': [2, 2, 2, 2, 1],
    'payload': ['endereco', 'umidade',
    ↪  'temperatura', 'gas', 'chamas'],
    'length': 9
}

PDU_RFID = {
    'attribute': ['tag'],
    'size': [8],
    'payload': ['tag'],
    'length': 9
}

PDU_TYPE = {
    b'\x11': PDU_ANA,
    b'\x12': PDU_RFID
}
```

6.3.3. conf.py

```
# --------------------------------
#     SERVER COFIGURATION
# --------------------------------
SERVER_HOST = '127.0.0.1'
SERVER_PORT = 3306

# --------------------------------
#     DEVICE CONTROL
# --------------------------------
DEVICE_INTERVAL = {
    'AAA8': 1,
    'AAAB': 2
}

# --------------------------------
#     SERIAL PORT COFIGURATION
# --------------------------------
SERIAL_READ = True
SERIAL_PORT = '/dev/ttyACM0'
SERIAL_BAUDRATE = 57600

# --------------------------------
#     DATABASE COFIGURATION
# --------------------------------
DB_HOST = 'localhost'
DB_NAME = 'lrg'
DB_USER = '123'
DB_PASSWORD = '1234'

# --------------------------------
#     DATABASE TABLE COFIGURATION
# --------------------------------
TB_NAME = 'acesso'
```

6.4. Dispositivo IoT

6.4.1. IoT.ino

```
#include <SPI.h>
#include <MFRC522.h>
```

```
#define SS_PIN 10
#define RST_PIN 9
MFRC522 mfrc522(SS_PIN, RST_PIN);   // Create
 ↪  MFRC522 instance.
#define RELE 7

int i = 0;
void setup()
{
  Serial.begin(57600);   // Inicia a serial
  SPI.begin();      // Inicia  SPI bus
  mfrc522.PCD_Init();   // Inicia MFRC522
  pinMode(RELE, OUTPUT); // Configura o rele
}

void loop()
{
  ler_cartao();
}


void ler_cartao(){
  if ( ! mfrc522.PICC_IsNewCardPresent()) // procura
   ↪  por novos cartoes
  {
    return;
  }

  if ( ! mfrc522.PICC_ReadCardSerial())  //
   ↪  seleciona um cartao
  {
    return;
  }

  Serial.write(0x12); // delimitador inicial
  for (int i = 0; i < 4; i++) {
        if (mfrc522.uid.uidByte[i]<16)
         ↪  Serial.print("0");
        Serial.print(mfrc522.uid.uidByte[i], HEX);
         ↪  // lendo dois bytes por vez e
         ↪  adicionando-os ao pacote
  }

   digitalWrite(RELE,HIGH);
   delay(2000);
   digitalWrite(RELE,LOW);

  Serial.println();
}
```

6.5. Simulação

Os arquivos CSV e ODS utilizados para a simulação podem ser encontrados em: *https://github.com/RNolioSC/TCC/releases/tag/1*.