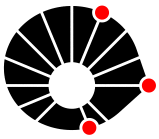

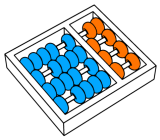

Plataforma Genérica de Suporte a Aplicações IoT Baseada em TV Boxes Reaproveitadas

T. G. Bannwart *L. F. G. Gonzalez* *Gustavo P. C. P. da Luz*
J. F. Borin

Relatório Técnico - IC-PFG-26-25
Projeto Final de Graduação
2026 - Julho

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Plataforma Genérica de Suporte a Aplicações IoT Baseada em TV Boxes Reaproveitadas

Tiago Godoi Bannwart [*] Luis Fernando Gomez Gonzalez [†]
Gustavo P. C. P. da Luz [‡] Juliana Freitag Borin [§]

Resumo

TV Boxes apreendidas pela Receita Federal, por falta de homologação da Anatel, preservam capacidade computacional que pode ser redirecionada para fins úteis, em vez de seguirem para destruição. Uma primeira versão de um sistema de suporte a aplicações IoT baseado nesse hardware reaproveitado já havia demonstrado, por meio de sua validação no sistema de estacionamento inteligente do Instituto de Computação da Unicamp, que uma TV Box é capaz de sustentar os serviços de retaguarda de uma aplicação IoT real, ainda que restrita às necessidades específicas desse caso de uso. Este trabalho apresenta o desenvolvimento de uma nova versão desse sistema, reprojetada como uma plataforma genérica e desacoplada de qualquer aplicação específica, capaz de oferecer observabilidade, monitoramento de disponibilidade orientado a eventos, atualização remota de firmware alinhada à RFC 9019, coleta de dados de sensores em tempo real e notificações extensíveis a qualquer dispositivo IoT que implemente o protocolo de comunicação definido. O sistema é composto por um *proxy* de autenticação baseado em TLS mútuo, um *broker* de mensagens MQTT e cinco serviços independentes, apoiados por uma camada de persistência em nuvem e acessíveis por meio de bibliotecas clientes voltadas a diferentes classes de dispositivo, de computadores de placa única a microcontroladores com recursos restritos. O sistema foi validado por meio da simulação do mesmo estacionamento inteligente utilizado como referência na versão anterior, permitindo observar, ao longo de duas semanas de operação contínua, estabilidade e recuperação adequada diante de falhas de dispositivo, de rede e de energia, além do funcionamento correto dos mecanismos de detecção de falhas e de atualização remota. Os resultados obtidos indicam que a nova versão preserva a capacidade já demonstrada da TV Box reaproveitada para essa função, ao mesmo tempo em que supera as limitações identificadas na versão anterior, ampliando o potencial de reaproveitamento desse hardware para qualquer projeto de IoT que necessite de serviços básicos de suporte.

Palavras-chave: Internet das Coisas; Computação na Névoa; TV Boxes; Economia Circular.

[*] t215386@dac.unicamp.br
[†] gonzalez@unicamp.br
[‡] g271582@dac.unicamp.br
[§] jufborin@unicamp.br

Sumário

1 Introdução

No Brasil, a fiscalização da venda de equipamentos eletrônicos sem homologação tem gerado, como efeito colateral, um volume crescente de hardware eletrônico apreendido cuja destinação padrão é o descarte. Esse cenário tem motivado tanto o poder público quanto a comunidade acadêmica a investigar formas de reaproveitamento funcional desses dispositivos como alternativa a essa prática. Entre janeiro de 2025 e janeiro de 2026, Anatel e Receita Federal retiraram de circulação mais de 1,3 milhão de produtos irregulares, movimentando um mercado avaliado em mais de R$ 136,6 milhões [1], número que se soma a um levantamento anterior de mais de 8 milhões de itens apreendidos por falta de homologação [2].

Entre os equipamentos mais recorrentes nessas apreensões estão as TV Boxes, dispositivos que prometem converter qualquer televisor em terminal de *streaming*, mas que costumam ser comercializados com acesso não autorizado a canais pagos e conteúdo sob demanda [2]. Contudo, o problema não se resume à irregularidade comercial já que, por não serem submetidas aos testes de segurança elétrica exigidos, e por eventualmente chegarem ao consumidor com *malware* pré-instalado, essas TV Boxes representam risco tanto para quem as utiliza quanto para a infraestrutura de telecomunicações à qual se conectam [3].

O destino tradicionalmente dado a esses equipamentos apreendidos é a destruição, alternativa que, além do custo logístico imposto ao Estado, converte um problema regulatório em passivo ambiental [4]. Como resposta, a Receita Federal tem direcionado parte desses lotes a instituições de ensino para fins de reaproveitamento [5], iniciativa que dialoga com os princípios da economia circular, modelo que propõe estender a vida útil de produtos eletrônicos como alternativa à extração de novos recursos e à geração de resíduo [6]. Com essa nova perspectiva, uma TV Box apreendida passa a ser um recurso computacional possivelmente valioso para diferentes aplicações, ao invés de um problema logístico a ser eliminado.

A viabilidade técnica do reaproveitamento já foi demonstrada experimentalmente em diferentes domínios de aplicação. da Luz *et al.* [4] avaliaram um *testbed* de 20 TV Boxes operando como nós de computação na borda em um cenário de contagem de pessoas por imagem, aplicável a cidades inteligentes, e observaram que os dispositivos sustentam cargas de inferência contínuas quando associados a modelos leves, superando inclusive equipamentos comerciais equivalentes em pegada de carbono quando considerada a matriz energética brasileira.

Em contexto agrícola, Gomes e Innocentini [7] reaproveitaram um receptor de TV Box, descaracterizado e reconfigurado como servidor do sistema supervisório de código aberto ScadaBR, integrado a um módulo cliente construído sobre o microcontrolador. Esse módulo reunia sensores de umidade e temperatura do solo e do ar e de vazão, além de um módulo de relés responsável por acionar solenoides de irrigação por gotejamento e microaspersão, e foi implantado em uma horta experimental. O trabalho evidencia que o hardware reaproveitado é suficiente até mesmo para hospedar um sistema supervisório completo, ainda que os próprios autores ressaltem a necessidade de adaptações físicas como a instalação de um *cooler* na carcaça do aparelho, para conter o superaquecimento decorrente da operação contínua.

Por outro lado, de Almeida *et al.* [8] avaliaram TV Boxes como nós de uma arquitetura de *Function-as-a-Service* (FaaS) distribuída ao longo de um contínuo computacional composto por camadas de sensoriamento, borda (*edge*) e névoa (*fog*), orquestrada pelo arcabouço *OpenFaaS*, nos dispositivos de borda, e *OpenFaaS Community Edition*, no servidor de névoa para executar funções de visão computacional, incluindo contagem de rostos via OpenCV e contagem de multidões via modelos YOLO. Comparando duas TV Boxes recondicionadas a duas versões de Raspberry Pi e a um servidor x86 de maior porte, os autores observaram que as TV Boxes obtiveram desempenho comparável, ou superior, ao de Raspberry Pis, e que os dispositivos de borda escalam de forma

eficiente sob requisições concorrentes, o que os torna adequados para cargas de trabalho menos exigentes dentro do contínuo computacional, ainda que o servidor de névoa permaneça necessário para as cargas mais pesadas.

Por fim, Silva *et al.* [9] utilizaram uma TV Box reconfigurada como hub de IoT de baixo custo voltado ao ensino de engenharia, integrando sensores e atuadores via protocolo MQTT (*Message Queuing Telemetry Transport*) a uma arquitetura modular composta por um motor de regras baseado em JSON (*JavaScript Object Notation*) e um módulo de processamento de linguagem natural, capaz de interpretar comandos de voz e texto por meio de um modelo de linguagem executado localmente ou de forma remota. Na validação, em que o hub controlava a temperatura interna de um rack de servidores por meio de um sensor e de ventoinhas acionadas via relé, o sistema manteve tempo médio de resposta dentro dos limites considerados aceitáveis para interação humano-computador. O trabalho reforça que TV Boxes reaproveitadas suportam não apenas cargas de processamento simples, mas também *pipelines* mais sofisticados, como a inferência de modelos de linguagem embarcados.

Em conjunto, esses resultados sustentam a hipótese central deste trabalho de que uma TV Box apreendida preserva capacidade computacional suficiente para operar, de forma contínua e confiável, como um nó de processamento útil fora do uso comercial original para o qual foi fabricada.

Nessa direção, foi proposto anteriormente o uso de uma TV Box reaproveitada como servidor de suporte a aplicações IoT, operando em regime de computação na névoa. A proposta foi validada por meio de um sistema de estacionamento inteligente implantado no Instituto de Computação da Unicamp, para o qual o dispositivo fornecia serviços como monitoramento de disponibilidade, atualização remota de firmware e coleta de métricas. O protótipo alcançou estabilidade operacional e baixo consumo energético, oferecendo evidência prática de que o hardware reaproveitado é suficiente para sustentar, de forma contínua, os serviços de apoio de uma aplicação IoT real [10].

Esse primeiro sistema, no entanto, foi projetado em função das necessidades específicas do estacionamento inteligente, e isso se refletiu em limitações que foram observadas conforme o sistema operou de forma contínua durante um período maior de tempo. Tais limitações, detalhadas na Seção 2 deste trabalho, evidenciavam a viabilidade do conceito, mas também revelavam um sistema construído em torno de um único caso de uso, cuja reutilização em outro domínio exigiria reescrita substancial.

Diante desse cenário, este trabalho apresenta o desenvolvimento de uma nova versão do sistema de suporte a aplicações IoT baseado em TV Boxes reaproveitadas, projetada não mais em torno de uma aplicação específica, mas como uma plataforma genérica, capaz de atender a qualquer dispositivo IoT que implemente um protocolo de comunicação padronizado. As seções seguintes descrevem o desenvolvimento do sistema proposto e discutem os resultados obtidos em sua validação, utilizando o sistema de estacionamento inteligente do Instituto de Computação da Unicamp como estudo de caso, de modo a permitir a comparação direta com a versão anterior.

2 Justificativa

Ao longo de sua operação contínua no sistema de estacionamento inteligente do Instituto de Computação da Unicamp, o primeiro sistema de suporte a aplicações IoT baseado em TV Box demonstrou, na prática e por um período prolongado, que o hardware reaproveitado é tecnicamente capaz de sustentar os serviços de apoio a uma aplicação IoT real. Contudo, apesar do sucesso, a operação prolongada acabou também expondo um conjunto de limitações estruturais, decorrentes de decisões de projeto tomadas em função das necessidades específicas do estacionamento inteligente no período do desenvolvimento.

O principal problema encontrado foi a falta de um mecanismo de *logging* para os dispositivos embarcados (no caso do estacionamento, o ESP8266). Isso significou que o diagnóstico de falhas era feito sem visibilidade sobre o que o dispositivo havia tentado executar, qual erro havia ocorrido ou em que momento, o que tornava a manutenção reativa e dependente de tentativa e erro, em vez de apoiada em uma análise sistemática de causa raiz.

Além disso, o monitoramento de disponibilidade dos dispositivos estava embutido no fluxo de dados da própria aplicação, uma vez que os *heartbeats* eram enviados como parte da chamada à API (*Application Programming Interface*) do sistema de vagas do estacionamento. Consequentemente, apenas os dispositivos que consumiam essa API específica podiam ser monitorados, o que impedia a extensão do mecanismo de monitoramento a qualquer outro tipo de aplicação IoT sem alterações estruturais no sistema.

As próprias métricas de operação do servidor, utilizadas para a avaliação experimental do sistema, eram coletadas de forma externa e manual, e não pelo sistema em operação. O servidor, portanto, não realizava autodiagnóstico de sua própria saúde da mesma forma que fazia para os demais dispositivos da rede.

Uma vez que os dados de saúde eram coletados, a verificação da disponibilidade dos dispositivos ocorria por sondagem ao banco de dados em períodos definidos. Esse intervalo implicava em uma janela de detecção de falhas de possivelmente vários minutos. Esse atraso para a percepção de uma falha real poderia ser uma limitação relevante em cenários que demandem resposta mais imediata a falhas.

Assim que uma falha era detectada, o sistema de notificação era acionado. Esse, por sua vez, estava restrito à plataforma Telegram, com um único destinatário e um conjunto fixo de dispositivos monitorados definidos diretamente no código-fonte. A inclusão de um novo destinatário, de uma nova plataforma de notificação ou de um novo dispositivo a ser monitorado exigia alteração e nova implantação do código, tornando a operação pouco flexível diante de múltiplos destinatários ou de um conjunto de dispositivos que muda com frequência.

No âmbito da manutenção do sistema, o mecanismo de atualização remota de firmware (OTA - *Over-the-Air*) era projetado para atender a um único dispositivo específico, e o servidor não recebia retorno dele sobre o andamento ou o resultado da atualização. Uma vez que o firmware era publicado, o sucesso ou a falha do processo permaneciam desconhecidos ao sistema.

Durante a operação do sistema, os dados coletados pelos sensores do estacionamento também não eram ingeridos diretamente pelo sistema. Informações sobre as vagas chegavam ao banco de dados enviados diretamente por um Raspberry Pi, e o totem de exibição para o usuário consumia essas informações periodicamente via API. A falta de um caminho direto e em tempo real implicava em latências para a apresentação dos dados mais novos, e várias consultas periódicas a API para buscar novos dados.

Ainda nesse quesito, não havia qualquer controle sobre a taxa de envio de dados pelos dispositivos. Um dispositivo defeituoso ou mal configurado poderia, em tese, inundar o banco de dados com mensagens, sem qualquer mecanismo de contenção ou de alerta automático sobre esse comportamento anômalo.

Por fim, e de modo mais fundamental, todas essas características foram implementadas em função das necessidades específicas do sistema de estacionamento inteligente para o qual o protótipo foi originalmente concebido, sem uma camada de abstração maior entre a aplicação e os serviços de suporte oferecidos pela TV Box. Reaproveitar o sistema para monitorar, atualizar ou coletar dados de qualquer outro tipo de dispositivo IoT exigiria, portanto, reescrita substancial de seus componentes.

Em conjunto, essas limitações delimitam o alcance do conceito original. Contudo, ao mostrar que uma única TV Box reaproveitada é capaz de operar continuamente como servidor de suporte a uma

aplicação IoT real, apesar de ser moldada por decisões de projeto específicas, foi percebido que essa capacidade foi explorada de forma restrita. Essas evidências justificam o desenvolvimento de uma nova versão do sistema que preserve o potencial já comprovado da TV Box reaproveitada para essa função, mas que a implemente de forma desacoplada de qualquer aplicação específica, observável, extensível e apta a atender a qualquer projeto ou dispositivo IoT, não apenas ao estacionamento inteligente que a originou.

3 Objetivos

Este trabalho tem como objetivo geral desenvolver uma nova versão do sistema de suporte a aplicações IoT baseado em TV Boxes reaproveitadas, pensada para funcionar de forma genérica, sem depender de nenhuma aplicação específica. Esse novo sistema deve ser capaz de oferecer observabilidade, monitoramento de saúde, atualização remota e coleta de dados para qualquer dispositivo IoT que implemente o protocolo de comunicação definido pelo sistema, além de notificações geradas a partir de eventos ocorridos nesses dispositivos, assim superando as limitações identificadas na versão anterior sem abrir mão da robustez e do baixo custo que caracterizam o uso desse hardware reaproveitado.

De forma específica, este trabalho tem como objetivos:

1. Prover observabilidade aos dispositivos, por meio da coleta estruturada de logs: com nível de severidade (por exemplo, debug, info e error), mensagem, *timestamp* e metadados; e com isso permitir diagnóstico de falhas baseado em análise de causa raiz em vez de inspeção manual.

2. Desacoplar o monitoramento de disponibilidade da lógica da aplicação específica que o utiliza, tornando-o aplicável a qualquer dispositivo que publique seu estado por meio do protocolo padronizado do sistema.

3. Substituir a detecção de falhas por sondagem periódica por um modelo orientado a eventos, capaz de identificar a ausência de sinais de vida dos dispositivos dentro de uma janela de tempo determinística e configurável, reduzindo a latência de detecção.

4. Generalizar o mecanismo de atualização remota de firmware (OTA) para múltiplos dispositivos simultaneamente, alinhando sua arquitetura à padronização proposta pela RFC 9019 e portanto, também incorporando rastreamento automático do resultado de cada atualização.

5. Tornar o sistema de notificação extensível e configurável, permitindo o cadastro dinâmico de destinatários, plataformas de notificação e dispositivos monitorados, de modo que a inclusão de novas plataformas de comunicação ou de novos dispositivos não exija reimplementar a lógica já existente, bastando estender as interfaces definidas pelo sistema.

6. Viabilizar a ingestão direta e em tempo real dos dados gerados por sensores de qualquer dispositivo IoT, eliminando intermediários e latências desnecessárias entre a coleta e o consumo deles por outros dispositivos da rede, mas mantendo seu armazenamento.

7. Incorporar mecanismos de proteção contra o envio excessivo de dados por dispositivos individuais, prevenindo sobrecarga do sistema de armazenamento e sinalizando automaticamente comportamentos anômalos.

8. Implementar o autodiagnóstico do próprio servidor, com coleta automática e contínua de métricas de uso de CPU, memória e temperatura, tratando o próprio nó como mais um dispositivo monitorado da rede.

9. Reprojetar o sistema como uma plataforma genérica, definida em torno de um protocolo de comunicação padronizado, de modo que qualquer dispositivo ou projeto IoT possa se beneficiar de seus serviços de suporte sem exigir modificações nos componentes centrais do sistema.

4 Desenvolvimento do Trabalho

4.1 Arquitetura Geral

O sistema desenvolvido neste trabalho é organizado em três camadas, seguindo o paradigma de *edge-fog-cloud continuum*. Na borda, encontram-se os dispositivos IoT, que podem ser microcontroladores e demais equipamentos embarcados, e que consomem os serviços do sistema por meio das bibliotecas cliente descritas na Seção 4.7. Na camada de névoa, executa-se o próprio sistema, hospedado em uma única TV Box reaproveitada. Ela atua como um servidor Linux que concentra um proxy reverso responsável pela autenticação de todo o tráfego recebido, um *broker* de mensagens MQTT e um conjunto de serviços responsáveis pelas funcionalidades de suporte oferecidas aos dispositivos. Na nuvem, por fim, encontram-se os serviços externos de persistência e visualização de dados, incluindo um banco de dados de séries temporais, um sistema de agregação de logs, um serviço de armazenamento de objetos e uma ferramenta de visualização, além dos canais de notificação (Telegram, e-mail, entre outros) para os quais os operadores do sistema são alertados.

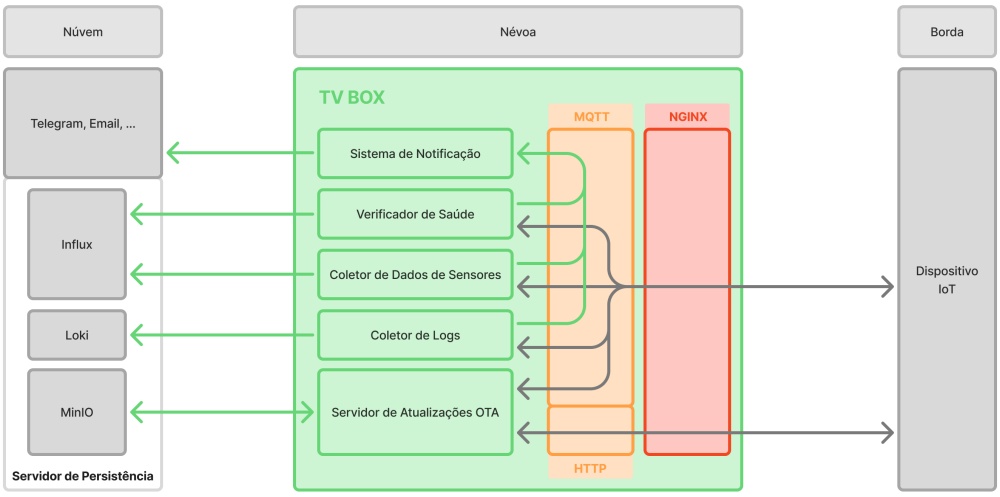

Figura 1: Arquitetura geral do sistema.

A Figura 1 ilustra a arquitetura geral da solução proposta. Nela, é possível observar que todo o tráfego originado nos dispositivos converge para o proxy reverso, responsável por autenticar a conexão antes de encaminhá-la a um dos cinco serviços do sistema: o sistema de notificação, o verificador de saúde, o coletor de dados de sensores, o coletor de logs e o servidor de atualização remota de firmware. Cada um desses serviços, por sua vez, persiste os dados que recebe em um dos serviços da camada de nuvem, e o sistema de notificação é o único ponto de saída em direção aos canais externos de alerta.

As subseções a seguir apresentam a configuração da TV Box utilizada e percorrem essa arquitetura da borda até a nuvem, detalhando o mecanismo de autenticação, o *broker* de mensagens, cada um dos serviços e, por fim, a camada de persistência.

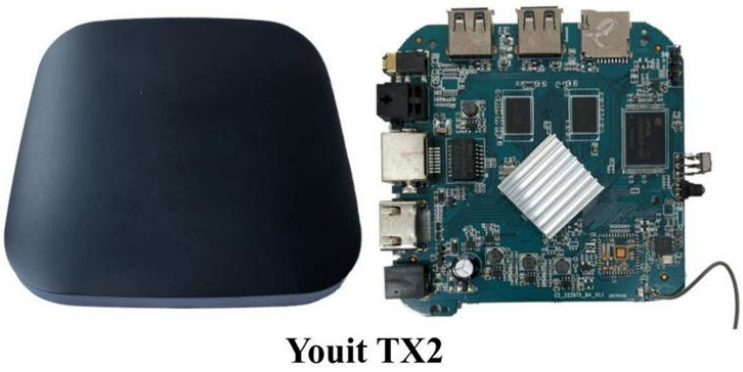

Figura 2: TV Box modelo Youit TX2 usada no trabalho.

4.2 Configuração da TV Box

O sistema foi implantado em uma TV Box do modelo Youit TX2, conforme representado na Figura 2, o mesmo modelo utilizado na primeira versão do sistema de suporte a aplicações IoT. Trata-se de um dispositivo de hardware limitado, equipado com uma CPU ARM Cortex-A7MP de 32 bits com quatro núcleos operando a 1,2 GHz, 2 GB de memória RAM LPDDR3 e 16 GB de armazenamento em memória flash eMMC, além de suporte a Ethernet, Wi-Fi e Bluetooth. Em termos de capacidade de processamento, esse modelo é comparável a um Raspberry Pi 3B [10].

O sistema operacional Android original do dispositivo, que viabiliza o acesso não autorizado a conteúdo que motiva a apreensão desses equipamentos, foi substituído por uma distribuição Linux. Para isso, foi adotado o Armbian versão 25.11.1 Bookworm Minimal, baseado no kernel Linux 6.12.58, uma distribuição baseada em Debian e otimizada para hardware ARM, de código aberto, personalizável e eficiente em termos de uso de recursos. Essa escolha se justifica tanto pela leveza da distribuição, que permite instalar apenas os pacotes necessários para a operação do sistema sem o peso de um ambiente de desktop, quanto pelo suporte específico a dispositivos baseados no *System-on-Chip* utilizado nesse modelo de TV Box.

4.3 Autenticação

Todo o acesso ao sistema, seja via MQTT, seja via HTTP, é intermediado por um proxy reverso baseado em NGINX[1], responsável por autenticar cada conexão antes de encaminhá-la aos serviços internos. A autenticação é feita exclusivamente por meio de TLS (*Transport Layer Security*) mútuo (mTLS): não há autenticação em nível de aplicação, como usuário e senha ou chaves de API. A única forma de estabelecer uma conexão com o sistema é apresentar um certificado de cliente X.509 válido, assinado por uma Autoridade Certificadora (CA) privada mantida no próprio servidor.

Essa CA privada é gerada localmente no servidor, e os certificados de cliente, que são constituídos por um par de chave privada e certificado assinado por essa CA, são igualmente gerados no servidor e distribuídos a cada dispositivo antes de sua entrada em operação.

Diferentemente do certificado de cliente, o certificado apresentado pelo próprio servidor ao estabelecer a conexão TLS é emitido de forma tradicional, por uma autoridade certificadora pública, no caso a Let's Encrypt, e não pela CA privada do sistema. Essa escolha atende a dois propósitos complementares: enquanto a CA privada garante que o servidor só aceita conexões de dispositivos previamente autorizados, o certificado público emitido pela Let's Encrypt garante ao dispositivo

[1] https://nginx.org/

que o servidor com o qual ele está se comunicando é de fato autêntico, e não um servidor malicioso se passando pelo sistema.

Essa camada de autenticação é particularmente relevante neste contexto porque dados potencialmente sensíveis trafegam pelo sistema, como por exemplo credenciais de rede Wi-Fi, tokens de acesso ou outras informações de configuração que podem estar embutidas nos próprios binários de firmware distribuídos pelo mecanismo de atualização remota. Garantir que toda a comunicação seja criptografada e mutuamente autenticada mitiga o risco de interceptação ou de acesso indevido a essas informações.

Nesse sistema, dois pontos de entrada são expostos: a porta 443, que expõe via HTTPS a interface REST (*Representational State Transfer*, um estilo arquitetural para APIs baseadas em HTTP) do servidor de atualização de firmware, e a porta 8883, que expõe o protocolo MQTT sobre TLS. Em ambos os casos, após a validação do certificado do cliente, o proxy encaminha a conexão já autenticada e decifrada ao serviço interno correspondente, que roda apenas na interface local do próprio servidor e não implementa autenticação própria. Essa responsabilidade é delegada inteiramente à camada de proxy.

4.4 *Broker* de Mensagens

A comunicação entre os dispositivos e os serviços do sistema, bem como a comunicação entre os próprios serviços, ocorre por meio de um *broker* MQTT baseado no Mosquitto[2].

Essa escolha se justifica, em primeiro lugar, pelo modelo de publicação/assinatura (*publish/subscribe*) do protocolo. Um dispositivo publica uma mensagem em um tópico sem se preocupar com quem irá consumi-la, e um serviço assina os tópicos de seu interesse sem se preocupar com qual dispositivo os publica. Essa separação é o que permite ao sistema atender a qualquer dispositivo ou serviço que adote o protocolo definido, sem exigir que um conheça previamente a existência do outro, sendo essa a característica que viabiliza a generalidade buscada por esta nova versão do sistema. Além disso, esse modelo permite que os dados sejam encaminhados diretamente a quem se interessa por eles assim que são publicados, o que representa uma melhoria em relação ao modelo de requisição-resposta empregado pelo sistema anterior, no qual os dispositivos precisavam realizar requisições periódicas ao servidor para verificar a existência de novos dados.

Outro fator que motiva essa escolha é o fato de o protocolo ser amplamente adotado no ecossistema IoT por sua baixa sobrecarga de cabeçalho, por suportar operação sobre conexões instáveis, já que possui filas de mensagens *offline* e níveis de qualidade de serviço configuráveis, e por permitir que mensagens fiquem retidas no *broker*, sendo entregues automaticamente a um novo assinante assim que ele se conecta. Esse último mecanismo é particularmente útil, por exemplo, para garantir que um dispositivo recém-conectado receba imediatamente a versão mais recente de firmware disponível destinado a ele, sem que seja necessário aguardar uma nova publicação.

Os tópicos do sistema seguem uma hierarquia comum, conforme ilustra a Figura 3. Essa hierarquia é detalhada a seguir, na descrição de cada serviço e dos canais que ele assina ou publica.

Todas essas mensagens transportam seus dados codificados em CBOR (*Concise Binary Object Representation*), um formato binário compacto escolhido em lugar do mais comumente utilizado JSON textual por gerar *payloads* menores, o que é muito vantajoso tanto para dispositivos embarcados com recursos de processamento e memória limitados quanto para conexões de rede de baixa largura de banda, cenário comum em aplicações IoT.

[2]https://mosquitto.org/

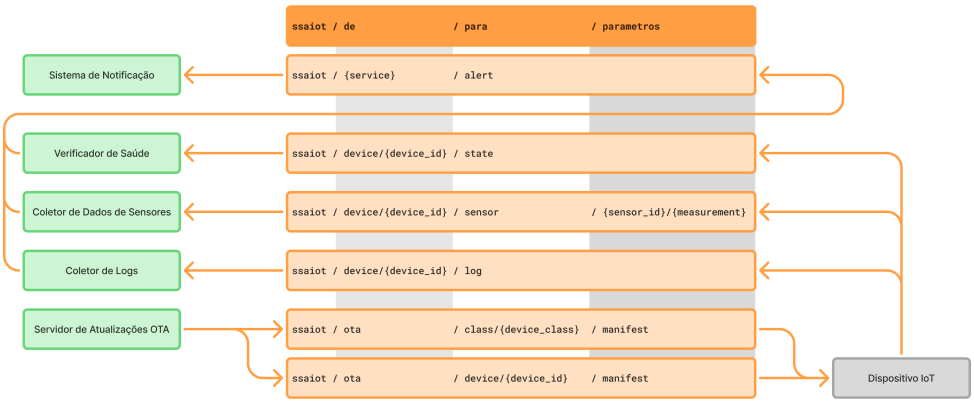

Figura 3: Organização dos tópicos do broker de mensagens.

4.5 Serviços

Os cinco serviços do sistema são implementados em Go e compilados como binários independentes, cada um executado como um processo próprio gerenciado pelo `systemd`, o que permite que sejam reiniciados ou atualizados individualmente, sem afetar a disponibilidade dos demais.

4.5.1 Servidor de Atualização OTA

A implementação desse serviço busca seguir os princípios definidos pela RFC 9019, que propõe uma arquitetura de referência para atualização de firmware em dispositivos IoT. Um dos princípios centrais dessa arquitetura é a separação entre o manifesto de uma atualização, um documento contendo metadados como a versão do firmware e mecanismos de verificação de integridade, e o binário do firmware propriamente dito, permitindo que um dispositivo valide a autenticidade e a integridade de uma atualização antes de instalá-la. Como base para essa implementação, foi utilizado o método proposto por Sousa [11] em sua dissertação de mestrado sobre atualizações OTA para dispositivos IoT.

De forma concreta, o objetivo desse serviço é receber, armazenar e distribuir novas versões de firmware para os dispositivos IoT. Para isso, cada dispositivo é caracterizado por uma classe (*device class*) e por um identificador (*device id*), o que permite que uma atualização seja direcionada tanto a um dispositivo específico quanto a todos os dispositivos de uma mesma classe de uma só vez.

O serviço expõe uma API HTTP, intermediada pelo NGINX, responsável tanto por receber (*upload*) quanto por disparar novas atualizações. Os endpoints dessa API estão resumidos na Tabela 1.

Ao disparar uma atualização, o serviço gera um manifesto, contendo a versão do firmware, sua URL de *download*, o *checksum* e a classe de dispositivo a que se destina, ou o identificador de um dispositivo específico, e o publica via MQTT, como mensagem retida, no tópico endereçado ao dispositivo específico ou à sua classe. A URL de *download* presente no manifesto é gerada pelo serviço de armazenamento de objetos, na camada de persistência, na forma de uma URL pré-assinada, mas é reescrita para que o *download* seja realizado através do proxy NGINX, e não diretamente nesse serviço, garantindo que apenas dispositivos autenticados por certificado mTLS

Tabela 1: Endpoints da API REST do servidor de atualização remota.

Método e rota	**Descrição**	**Corpo da requisição**
`GET /firmware`	Lista os firmwares disponíveis, agrupados por classe de dispositivo	Sem corpo
`POST /firmware`	Envia um novo firmware	Multipartes: binário, versão e classe de dispositivo
`GET /firmware/:id`	Dados de um firmware específico (versão, checksum, data da última modificação)	Sem corpo
`DELETE /firmware/:id`	Remove um firmware específico do armazenamento	Sem corpo
`POST /firmware/update/:id`	Dispara a atualização de um firmware específico	JSON: classe ou identificador do destinatário

tenham acesso ao binário do firmware.

Do lado do dispositivo, a biblioteca cliente correspondente recebe o manifesto, valida se a classe do dispositivo corresponde à do manifesto, baixa o binário, verifica seu *checksum*, aciona uma função de *callback* fornecida pela aplicação para efetivamente aplicar a atualização, e registra cada uma dessas etapas por meio do serviço de *log*, inclusive em caso de erro. Esse fluxo segue os conceitos gerais de arquitetura de atualização de firmware propostos pela RFC 9019, atendendo ao objetivo de alinhar o mecanismo de OTA a uma padronização reconhecida pela comunidade de IoT. No caso dos microcontroladores ESP8266 e ESP32, não há uma função de *callback* fornecida pela aplicação: a biblioteca cliente aciona diretamente as funções de atualização expostas pelas próprias bibliotecas da Espressif.

4.5.2 Coletor de Dados de Sensores

Esse serviço recebe as leituras de sensores publicadas pelos dispositivos e as armazena como dados de série temporal no banco de dados de séries temporais. Ele assina o tópico curinga `ssaiot/device/+/sensor/+/+`, no qual o primeiro curinga corresponde ao identificador do dispositivo publicante, o segundo ao identificador do sensor específico, e o último segmento do tópico indica o tipo de grandeza medida (*measurement*), como temperatura ou umidade. O valor medido e, opcionalmente, um *timestamp* são transportados no *payload* da mensagem. Quando o *timestamp* está ausente, ele é definido pelo horário do próprio servidor no momento do recebimento.

Para evitar que um dispositivo defeituoso ou mal configurado sobrecarregue o banco de dados, o serviço aplica um limite de cinco mensagens por minuto para cada combinação de dispositivo e sensor, controlado por um mapa em memória protegido por exclusão mútua e reiniciado a cada minuto por uma rotina em segundo plano. Mensagens que excedem esse limite são descartadas, e um alerta é publicado automaticamente, que será encaminhado ao operador do sistema pelo serviço de notificação, identificando o dispositivo e o sensor responsáveis pelo comportamento anômalo.

4.5.3 Coletor de Logs

Esse serviço recebe mensagens de log estruturadas, com nível (informação, depuração ou erro), texto e metadados arbitrários, publicadas pelos dispositivos no tópico `ssaiot/device/{device_id}/log`, e as persiste no Loki, o sistema de agregação de logs da camada de nuvem, por meio de sua API HTTP de ingestão. No Loki, cada registro se torna um *stream* identificado pelos metadados do próprio log somados ao identificador do dispositivo e ao nível da mensagem, permitindo consultas filtradas por qualquer uma dessas dimensões. Assim como no servidor de atualização remota, esse serviço é implementado em torno de interfaces e abstrações, o que permite substituir o Loki por

outro agregador de logs ou mesmo por um banco de dados distinto sem necessidade de alterar sua lógica interna.

Esse serviço desempenha também um papel adicional: caso um log de nível de erro contenha uma marcação relacionada a atualização de firmware, ele gera automaticamente um alerta descrevendo a versão do firmware e a exceção associada, publicando-o para o sistema de notificação. Isso fecha o ciclo de rastreamento do processo de OTA iniciado pelo servidor de atualização remota.

4.5.4 Verificador de Saúde

Esse serviço monitora a disponibilidade dos dispositivos a partir dos heartbeats que eles publicam periodicamente. Um dispositivo se registra para monitoramento publicando no tópico `ssaiot/device/{device_id}/state/monitor`, informando o intervalo de tempo (timeout) após o qual, na ausência de novos heartbeats, deve ser considerado offline. Esse registro é persistido em disco de forma atômica, de modo a sobreviver a reinícios do próprio serviço.

A cada heartbeat recebido no tópico `ssaiot/device/{device_id}/state`, o serviço atualiza o horário do último sinal de vida do dispositivo e grava o estado reportado no banco de dados de séries temporais. Caso o dispositivo estivesse marcado como *offline*, um alerta de "dispositivo online" é publicado, encaminhado ao operador também através do serviço de notificação.

A verificação de disponibilidade em si é feita por uma rotina que roda a cada segundo, comparando o horário atual com o horário do último *heartbeat* de cada dispositivo monitorado. Caso o intervalo configurado seja ultrapassado sem um novo sinal de vida, o dispositivo é marcado como *offline* e um alerta correspondente é publicado. Esse mecanismo substitui o antigo modelo de sondagem periódica por um processo cuja janela de detecção é determinística e configurável por dispositivo.

Além do monitoramento dos dispositivos externos, esse mesmo serviço realiza a coleta periódica de métricas do próprio servidor, como uso de CPU, memória e temperatura, a cada 30 segundos, tratando o próprio nó como mais um dispositivo monitorado da rede.

4.5.5 Sistema de Notificação

Esse serviço assina o tópico curinga `ssaiot/+/alert`, recebendo assim, em um único ponto, todos os alertas gerados pelos demais serviços: de disponibilidade, de erro de atualização de firmware ou de violação do limite de envio de dados de sensores. Em vez de encaminhar esses alertas a um destino fixo, o serviço mantém um registro de destinatários cadastrados dinamicamente, cada um associado a uma plataforma de notificação, a um conjunto de tipos de alerta que deseja receber e a um conjunto de dispositivos que deseja monitorar. Ao receber um alerta, o serviço verifica quais destinatários cadastrados correspondem a ele, evitando notificar a mesma plataforma mais de uma vez para o mesmo evento.

A plataforma de notificação atualmente integrada é o Telegram, por meio de sua API de bot. Ampliar esse conjunto de plataformas ou cadastrar novos destinatários não exige alterar a lógica de despacho de alertas, bastando implementar a interface de plataforma já definida pelo serviço e realizar o cadastro correspondente no registro de destinatários.

4.6 Camada de Persistência

A camada de nuvem reúne os serviços externos responsáveis por agregar, armazenar e expor os dados coletados pelo sistema.

O InfluxDB, um banco de dados de séries temporais, armazena tanto as leituras de sensores (*bucket* dedicado, com o identificador do dispositivo e do sensor como *tags*) quanto os estados

reportados nos *heartbeats* (*bucket* separado, com o identificador do dispositivo como *tag*). A escolha de um banco de séries temporais se justifica pela própria natureza dos dados produzidos pelo sistema, já que leituras de sensores e *heartbeats* são séries de valores associados a instantes de tempo, o que favorece tanto a eficiência de escrita e armazenamento quanto consultas agregadas por janelas de tempo.

O Loki agrega os logs estruturados enviados pelo coletor de logs, indexando-os por rótulos (dispositivo, nível, e demais metadados) em vez de indexação textual completa. Essa escolha auxilia a análise e a busca de logs por parte dos operadores do sistema, além de se integrar nativamente com a mesma ferramenta de visualização usada para os dados do InfluxDB.

O MinIO, um serviço de armazenamento de objetos compatível com a API do S3, guarda os binários de firmware distribuídos pelo servidor de atualização remota, organizados por classe de dispositivo e versão, com o *checksum* do arquivo salvo como metadado. Diferentemente de dados de sensores ou de logs, binários de firmware são objetos não estruturados e potencialmente grandes, para os quais um serviço de armazenamento de objetos é mais adequado do que um banco de séries temporais ou um sistema de agregação de logs. Além disso, por ser compatível com a API do S3, esse serviço também permite reaproveitar bibliotecas e ferramentas amplamente utilizadas em ambientes de nuvem.

Por fim, o Grafana funciona como a camada de visualização do sistema, conectando-se tanto ao InfluxDB quanto ao Loki como fontes de dados, e permitindo observar em um único painel tanto as séries temporais de sensores e de saúde dos dispositivos quanto seus logs, sem que o sistema precise implementar sua própria interface de visualização. O plugin desenvolvido para o Grafana, descrito na seção seguinte, complementa essa camada ao também implementar a API do servidor de atualização remota, permitindo que o gerenciamento de atualizações de firmware seja realizado na mesma plataforma usada para observação de métricas e logs, de forma abrangente e centralizada.

Cabe notar que, por os serviços do sistema serem implementados em torno de interfaces e abstrações que isolam a lógica da aplicação dos mecanismos concretos de armazenamento, é simples substituir qualquer um desses serviços de agregação e persistência por outros que façam mais sentido em contextos específicos de implantação, sem necessidade de alterar a lógica interna dos serviços.

4.7 Bibliotecas Cliente e Ferramentas de Acesso

Para que dispositivos de diferentes naturezas se conectem ao sistema sem que cada equipe de desenvolvimento precise reimplementar os detalhes do protocolo, como a autenticação mTLS, a codificação CBOR e a nomenclatura de tópicos, foram desenvolvidas três bibliotecas de cliente. Os três atuam de forma equivalente e oferecem os mesmos quatro serviços de mais alto nível já apresentados na descrição dos serviços do sistema (sensor, saúde, log e atualização de firmware), adaptando sua implementação ao contexto da linguagem e da classe de dispositivo a que se destinam.

As bibliotecas em Python e em Go atendem à mesma classe de dispositivos, ou seja, aqueles equipados com um sistema operacional completo e recursos computacionais mais amplos, como computadores de placa única, a exemplo do Raspberry Pi utilizado na validação deste trabalho. Cada um tem uma vantagem distinta. A biblioteca em Python se beneficia da simplicidade da linguagem e de sua ampla adoção, o que pode facilitar a integração. Já a biblioteca em Go tem como vantagem ser mais leve e gerar um binário único, já com todas as suas dependências, o que facilita a distribuição da aplicação cliente em diferentes dispositivos.

Por fim, foi escrita uma terceira biblioteca cliente, voltada a microcontroladores ESP8266 e ESP32 sob o framework Arduino, que são dispositivos com recursos de processamento e memória muito mais restritos que os anteriores. Por conta dessas restrições, foram necessárias diversas otimizações ao longo da implementação, como o dimensionamento cuidadoso dos buffers usados na

comunicação segura e a alocação dos certificados uma única vez, sem nunca liberá-los, entre outros ajustes voltados a preservar a pouca memória disponível nesses dispositivos.

A autenticação mTLS é implementada com bibliotecas padrão do ecossistema Arduino e Espressif, responsáveis por carregar a CA, o certificado de cliente e a chave privada em cada dispositivo. Já a comunicação MQTT utiliza a biblioteca PubSubClient, envolvida por um adaptador que adiciona reconexão automática para maior robustez, suporte a curingas na assinatura de tópicos e múltiplos callbacks por tópico, enquanto a codificação e a decodificação das mensagens em CBOR são feitas por uma porta para Arduino da biblioteca TinyCBOR, encapsulada por uma camada própria que expõe construtores e funções de decodificação para cada tipo de mensagem trafegada.

O fluxo de atualização remota de firmware nessa biblioteca é síncrono, o que faz com que o laço principal do Arduino permaneça bloqueado durante todo o download. Ao receber a notificação de uma atualização, o dispositivo decodifica o manifesto e inicia o download do binário via HTTPS, lendo o conteúdo em blocos de 4096 bytes, de modo a se adequar à memória limitada desses microcontroladores. Cada bloco recebido é imediatamente gravado na partição de destino na *flash*, enquanto o *checksum* SHA-256 do arquivo é calculado durante o próprio download. Ao final, o checksum calculado é comparado ao informado no manifesto, e uma divergência interrompe e descarta a atualização. Concluída a gravação com sucesso, o dispositivo grava em seu armazenamento persistente a nova versão do firmware e uma marcação indicando que está prestes a reiniciar por causa de uma atualização, e então reinicia. No *boot* seguinte, essa marcação é detectada e removida e, no caso do ESP32, que conta com suporte nativo a *rollback* automático de firmware, o dispositivo também sinaliza ao *bootloader* que a nova versão foi validada com sucesso, evitando que essa proteção desfaça a atualização recém-aplicada.

Além das três bibliotecas clientes, foi desenvolvido um plugin para o Grafana que complementa a camada de persistência ao expor, na própria interface de visualização do sistema, uma forma de gerenciar atualizações de firmware. A Figura 4 mostra a aba de OTA desse plugin, organizada em três painéis que espelham diretamente alguns dos endpoints da API do servidor de atualização remota: um painel de envio de novos firmwares, no qual se informa a classe de dispositivo, a versão e o arquivo binário, um painel de navegação pelos firmwares já armazenados, organizados por classe de dispositivo, e um painel de disparo de atualizações, no qual o operador informa o identificador do firmware a ser distribuído e escolhe se o destinatário é uma classe inteira de dispositivos ou um dispositivo específico. Dessa forma, o gerenciamento completo do ciclo de vida de um firmware, do envio à distribuição, pode ser realizado na mesma ferramenta já utilizada para observar métricas, *heartbeats* e logs dos dispositivos, sem a necessidade de uma interface separada.

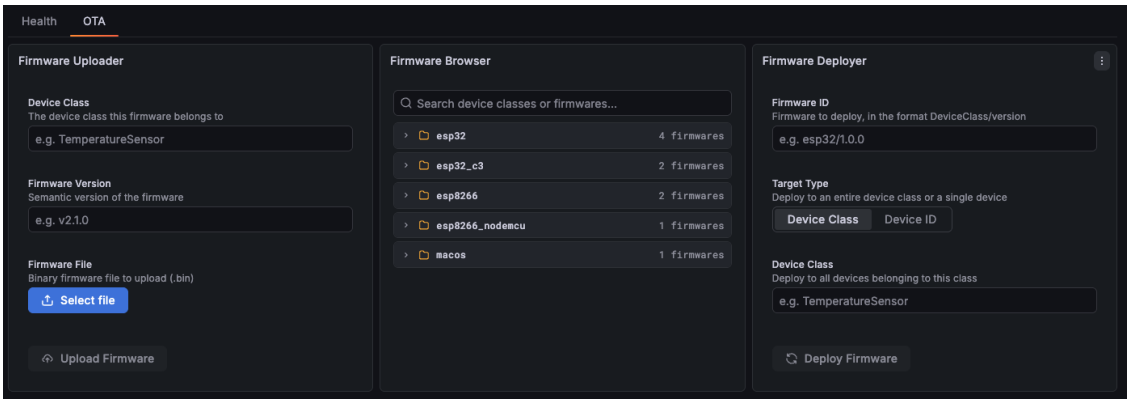

Figura 4: Aba de gerenciamento de OTA do plugin desenvolvido para o Grafana.

4.8 Validação Experimental

O sistema foi validado por meio da simulação do estacionamento inteligente do Instituto de Computação da Unicamp, cenário que originalmente motivou o desenvolvimento da primeira versão do sistema de suporte a aplicações IoT. Em relação ao funcionamento do sistema do estacionamento, um Raspberry Pi conectado a uma câmera captura imagens aéreas do estacionamento e, por meio de modelos de aprendizado profundo, estima o número de vagas disponíveis. Esse valor é, então, consumido por um totem de exibição, controlado por um microcontrolador ESP8266, que o apresenta em um painel de LEDs na entrada do estacionamento. Mais detalhes de como todo esse sistema foi implementado podem ser encontrados em Da Luz et al. [12].

Sob uma perspectiva do continuum computacional, a câmera acoplada ao Raspberry Pi e o ESP8266 do totem atuam como dispositivos de borda, enquanto o InfluxDB, hospedado nos servidores do Instituto de Computação, atua na nuvem. O servidor de suporte, tanto em sua versão original quanto na nova versão aqui proposta, atua na névoa, fazendo a ponte entre a borda e a nuvem.

A validação deste trabalho consistiu em simular todas as etapas que ocorrem no sistema de estacionamento atualmente em operação, buscando reproduzir resultados práticos equivalentes aos do sistema real. Isto é, buscou-se demonstrar que, dado que o novo sistema funciona corretamente, ele pode substituir o sistema de suporte atualmente em uso sem alterar o comportamento observável do estacionamento, com a mesma contagem de vagas apresentada ao usuário, mas apoiado por um sistema de suporte, coleta de dados e observabilidade mais robusto e capaz.

Para realizar essa simulação, foi utilizado um Raspberry Pi do mesmo modelo empregado atualmente no estacionamento, responsável por executar toda a inferência sobre um conjunto de fotos reais do estacionamento. Esse dispositivo utiliza todos os serviços oferecidos pelo servidor de suporte: o Health Checker, ao qual envia *heartbeats* periódicos, o Sensor Data Collector, para o qual publica os dados de vagas disponíveis a serem coletados e armazenados, o Log Collector, que permite observar seu funcionamento, e o OTA Server, empregado na atualização remota de arquivos do sistema.

Além do Raspberry Pi, dois microcontroladores, um ESP32 (modelo C3 SuperMini) e uma placa de desenvolvimento ESP8266, simulam o funcionamento do totem. Ambos escutam o tópico de dados de sensores para obter os valores de vagas disponíveis, enviam *heartbeats* ao Health Checker, escutam atualizações de firmware disparadas pelo OTA Server e enviam logs relativos ao seu próprio funcionamento.

Os cenários testados nesse ambiente de validação foram o desligamento do Raspberry Pi, com a expectativa de gerar uma notificação de alerta, o desligamento de cada um dos microcontroladores, com a mesma expectativa, a atualização remota de firmware dos microcontroladores, e a atualização, via OTA, de um arquivo que o Raspberry Pi usa durante o processo de inferência. Esses cenários abrangem, em conjunto, todos os serviços implementados pelo sistema, permitindo verificar seu funcionamento de ponta a ponta. Todo esse processo pôde ser acompanhado por meio do painel do Grafana.

5 Resultados

5.1 Operação Contínua do Estacionamento Simulado

Ao longo das duas semanas de simulação contínua do estacionamento inteligente, descrita na seção de Validação Experimental, o sistema permaneceu estável, sem apresentar falhas que exigissem intervenção manual ou reinício de qualquer um de seus serviços. O Raspberry Pi responsável pela

inferência sobre as imagens do estacionamento manteve-se conectado e publicando dados de forma consistente ao longo de todo o período, assim como os dois microcontroladores que simulavam o totem de exibição. Durante esse período, o ambiente de teste também passou por quedas de energia e de conectividade de rede não planejadas, de forma acidental, e todas as partes do sistema se recuperaram normalmente desses eventos, sem qualquer intervenção manual.

Por meio do painel do Grafana, foi possível acompanhar, durante toda a simulação, os *heartbeats* dos três dispositivos simulados, os dados de vagas publicados pelo Raspberry Pi e persistidos no InfluxDB, e os logs estruturados gerados por cada dispositivo, sem indícios de perda de mensagens ou de lacunas na coleta de dados. A Figura 5 apresenta um trecho desse painel, mostrando os indicadores de saúde do servidor de suporte e da câmera do Raspberry Pi permanecendo estáveis ao longo do intervalo exibido, além dos logs de *status* das vagas do estacionamento sendo recebidos de forma regular.

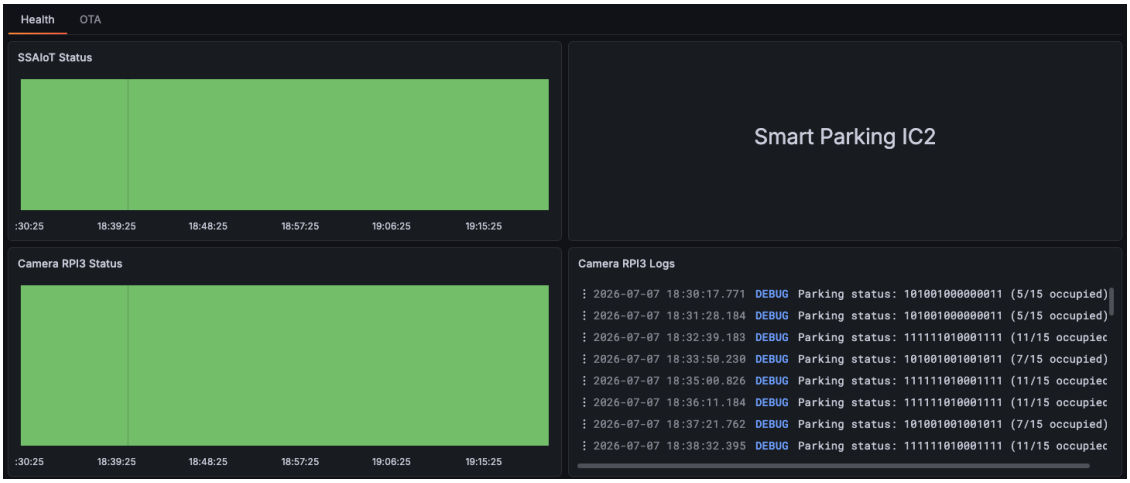

Figura 5: Painel do Grafana durante um trecho da operação contínua do estacionamento simulado.

5.2 Cenários de Falha e de Atualização Remota

Além da operação contínua, foram avaliados os quatro cenários descritos na metodologia de validação, todos com resultado positivo.

No cenário de desligamento do Raspberry Pi, o Health Checker deixou de receber *heartbeats* do dispositivo e, decorrido o intervalo de *timeout* configurado, marcou-o corretamente como *offline*, disparando um alerta que foi entregue ao operador do sistema por meio do Telegram pelo sistema de notificação. O mesmo comportamento foi observado ao desligar cada um dos microcontroladores que simulavam o totem, tanto o ESP32 quanto o ESP8266, confirmando que o mecanismo de detecção de falhas funciona de forma equivalente para os diferentes tipos de dispositivo suportados pelo sistema. A Figura 6 mostra um exemplo desse ciclo de alertas recebido no Telegram, com a notificação de desligamento, informando o intervalo de *timeout* configurado, seguida da notificação de retorno do dispositivo, informando havia quanto tempo ele esteve *offline*.

Nos cenários de atualização remota de firmware, tanto os microcontroladores quanto o Raspberry Pi receberam e aplicaram corretamente as atualizações disparadas pelo servidor de atualização remota. Os microcontroladores baixaram e instalaram um novo firmware sem intervenção manual, reiniciando ao final do processo, enquanto o Raspberry Pi recebeu com sucesso a atualização de um arquivo utilizado durante seu processo de inferência sobre as imagens do estacionamento. Em

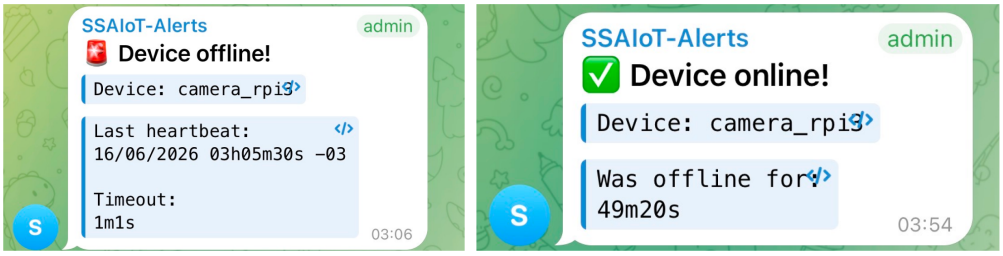

Figura 6: Alertas de dispositivo offline e de retorno ao estado online, recebidos via Telegram.

ambos os casos, o processo pôde ser acompanhado pelos logs estruturados publicados por cada dispositivo, sem registro de erros durante o download, a verificação de integridade ou a aplicação da atualização.

Como a biblioteca cliente dos microcontroladores implementa esse fluxo de forma síncrona, conforme descrito na Seção de Desenvolvimento, o dispositivo deixa de enviar *heartbeats* durante a etapa de download do firmware, o que fica visível no painel do Grafana como uma breve interrupção no indicador de saúde do dispositivo, seguida de sua recuperação assim que a atualização é concluída e o dispositivo reinicia. A Figura 7 ilustra esse comportamento durante a atualização do ESP8266. Especificamente no caso do ESP32 (C3 SuperMini), cuja placa utilizada possui uma antena mais sensível, foram observadas ocasionalmente falhas durante o download do firmware, mas os testes mostraram que o processo de atualização é concluído com sucesso dentro de, no máximo, três tentativas.

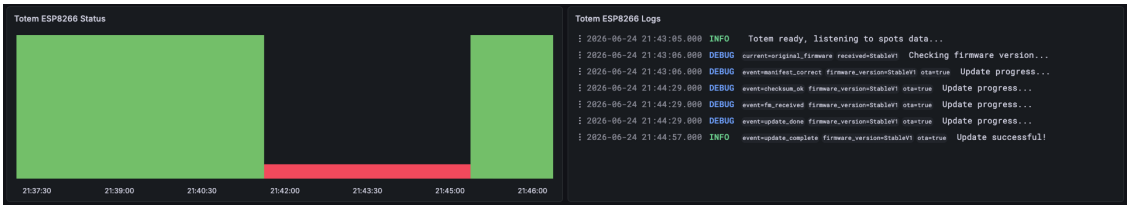

Figura 7: Painel do Grafana mostrando a interrupção de heartbeats durante a atualização de firmware do ESP8266, e os logs estruturados registrando cada etapa do processo.

5.3 Síntese dos Resultados

Em conjunto, os resultados obtidos indicam que a nova versão do sistema é capaz de sustentar o funcionamento do estacionamento inteligente nas mesmas condições que a versão anterior, ao mesmo tempo em que exercita, por meio dos quatro cenários testados, todos os serviços que a compõem. A operação estável ao longo de duas semanas, sob um padrão de uso semelhante ao da implantação real, incluindo sua recuperação diante de quedas de energia e de rede não planejadas, e a resposta correta a cada um dos cenários de falha e de atualização remota, constituem evidência de que o sistema atende às limitações apresentadas na Justificativa deste trabalho.

6 Conclusão

Este trabalho partiu do pressuposto de que TV Boxes apreendidas pela Receita Federal, por não possuírem homologação da Anatel, possuem capacidade computacional que pode ser redirecionada para fins úteis, em lugar de seguirem para destruição. Uma primeira versão de um sistema de suporte a aplicações IoT baseado nesse hardware reaproveitado já havia demonstrado, na prática e por um período prolongado, que uma TV Box é capaz de sustentar tais serviços de uma aplicação IoT real, validada por meio do sistema de estacionamento inteligente do Instituto de Computação da Unicamp [10]. Essa mesma operação prolongada, no entanto, expôs um conjunto de limitações estruturais, decorrentes de decisões de projeto tomadas em função das necessidades específicas daquele caso de uso, que restringiam a reutilização do sistema em qualquer outro domínio de aplicação.

Diante desse cenário, este trabalho apresentou o desenvolvimento de uma nova versão desse sistema, reprojetada como uma plataforma genérica e desacoplada de qualquer aplicação específica. O sistema resultante é composto por um proxy de autenticação baseado em TLS mútuo, um *broker* de mensagens MQTT e cinco serviços independentes, responsáveis por observabilidade, monitoramento de disponibilidade orientado a eventos, atualização remota de firmware alinhada à RFC 9019, coleta de dados de sensores em tempo real e notificações extensíveis, todos apoiados por uma camada de persistência em nuvem e acessíveis por meio de bibliotecas cliente voltados a diferentes classes de dispositivo, de computadores de placa única a microcontroladores com recursos bastante restritos.

A validação do sistema, conduzida por meio da simulação do mesmo estacionamento inteligente que serviu de referência para a versão anterior, permitiu observar, de forma qualitativa, que o sistema se manteve estável ao longo de duas semanas de operação contínua, inclusive diante de quedas de energia e de rede não planejadas, e respondeu corretamente aos cenários de falha e de atualização remota exercitados. Esses resultados indicam que os objetivos estabelecidos para este trabalho foram alcançados: a nova versão preserva a capacidade já demonstrada da TV Box reaproveitada para atuar como servidor de suporte a aplicações IoT, ao mesmo tempo em que supera as limitações identificadas na versão anterior, sem exigir hardware adicional ou abrir mão dos fatores que motivam o reaproveitamento desses dispositivos.

A principal contribuição deste trabalho é, portanto, mostrar que a viabilidade de reaproveitar TV Boxes apreendidas como servidores de suporte a IoT não precisa ficar restrita a uma única aplicação. Ao reprojetar o sistema em torno de um protocolo de comunicação padronizado e de uma arquitetura extensível, este trabalho amplia o potencial de reaproveitamento desse hardware para qualquer projeto de IoT que necessite de serviços básicos de observabilidade, monitoramento, atualização remota e coleta de dados, reforçando o papel desses dispositivos apreendidos como unidades computacionais úteis, e não apenas como dispositivos a serem descartados.

6.1 Limitações e Trabalhos Futuros

A validação realizada neste trabalho teve caráter predominantemente qualitativo. Não foram coletadas métricas quantitativas de desempenho, como uso de CPU, memória, temperatura e consumo de energia, de forma sistemática ao longo do período de teste, nem métricas relacionadas ao uso da rede, como o volume de pacotes trafegados entre os dispositivos e o servidor de suporte. A coleta de métricas quantitativas equivalentes para esta nova versão, e sua comparação direta com a versão anterior, ficam como sugestão para trabalhos futuros.

Embora a autenticação mútua garanta que apenas dispositivos portadores de um certificado válido consigam se conectar ao sistema, ela não implementa, atualmente, um esquema de auto-

rização que restrinja os tópicos que cada dispositivo pode publicar ou assinar de acordo com sua própria identidade. Um dispositivo mal configurado, ou cujo certificado tenha sido reaproveitado indevidamente, poderia publicar em um tópico pertencente a outro dispositivo e se passar por ele perante os demais serviços do sistema. A introdução de um mecanismo de autorização por tópico, vinculado à identidade presente no certificado de cada dispositivo, é apontada como um trabalho futuro relevante para fechar essa lacuna de segurança.

Este trabalho também se limitou à validação do sistema por meio de uma simulação do estacionamento inteligente, e não à sua implantação na instalação real em operação no Instituto de Computação da Unicamp. A substituição efetiva do sistema de suporte atualmente em uso pela versão aqui desenvolvida, e a observação de seu comportamento em produção ao longo de um período extenso, permanecem como um passo natural para trabalhos futuros.

Por fim, durante o desenvolvimento surgiu a ideia de que o próprio servidor de atualização remota pudesse validar um firmware logo após o seu envio, executando-o em uma simulação do dispositivo alvo, por exemplo em uma máquina virtual ou em um ambiente de emulação equivalente, antes de disponibilizá-lo para distribuição. Um mecanismo desse tipo permitiria diagnosticar problemas em um novo firmware de forma antecipada, reduzindo o risco de uma atualização malsucedida ser aplicada a dispositivos já em operação. Essa possibilidade não chegou a ser implementada neste trabalho, mas fica registrada como sugestão para trabalhos futuros.

7 Agradecimentos

À minha família, os de sangue e os que escolhi, por prover tudo que precisei ao longo dos estudos, pelo incentivo constante durante toda essa jornada e pelo suporte incondicional em cada etapa.

Aos meus amigos, pela companhia nas partes boas e difíceis da graduação, e por tornarem esses anos tão especiais.

Aos colegas do grupo de pesquisa do Laboratório Discovery, Gustavo da Luz, Luis Gonzalez e Gabriel Sato, por terem me acolhido e auxiliado nas pesquisas. Em especial, agradeço à minha orientadora, Professora Dra. Juliana Freitag Borin, por ter me recebido no laboratório e me orientado ao longo de um ano e meio.

Ao Instituto de Computação (IC) e à Universidade Estadual de Campinas (UNICAMP), e a toda a sua comunidade, pela excelência na minha formação acadêmica e por tanto terem contribuído para minha evolução pessoal.

Referências

[1] Agência Gov. “Combate à pirataria: Anatel retira mais de 1,3 milhão de produtos do mercado entre 2025 e 2026,” acesso em 19 de jun. de 2026. endereço: `https://agenciagov.ebc.com.br/noticias/202604/anatel-retira-mais-de-1-3-milhao-de-produtos-irregulares-do-mercado-entre-2025-e-2026`.

[2] Ministério das Comunicações. “Anatel retira de circulação mais de 8 milhões de produtos de telecomunicações por falta de homologação, correspondendo a R$ 833,6 milhões,” acesso em 19 de jun. de 2026. endereço: `https://www.gov.br/mcom/pt-br/noticias/2025/novembro/anatel-retira-de-circulacao-mais-de-8-milhoes-de-produtos-de-telecomunicacoes-por-falta-de-homologacao-correspondendo-a-r-833-6-milhoes`.

[3] Agência Nacional de Telecomunicações. "Anatel lança painel com informações sobre bloqueios administrativos de TV Boxes ilegais." português, Agência Nacional de Telecomunicações, acesso em 19 de jun. de 2026. endereço: https://www.gov.br/anatel/pt-br/assuntos/noticias/anatel-lanca-painel-com-informacoes-sobre-bloqueios-administrativos-de-tv-boxes-ilegais.

[4] G. P. C. P. da Luz, G. M. Sato, L. F. G. Gonzalez e J. F. Borin, "Repurposing of TV boxes for a circular economy in smart cities applications," en, *Scientific Reports*, v. 15, n. 1, p. 22 638, 2025, ISSN: 2045-2322. DOI: 10.1038/s41598-025-97379-4.

[5] Secretaria Especial da Receita Federal do Brasil. "Receita Federal em Brasília destina 30 mil aparelhos de TV Box para serem transformados em minicomputadores," acesso em 19 de jun. de 2026. endereço: https://www.gov.br/receitafederal/pt-br/assuntos/noticias/2024/dezembro/receita-federal-em-brasilia-destina-30-mil-aparelhos-de-tv-box-para-serem-transformados-em-minicomputadores.

[6] T. S. Gaur, V. Yadav, S. Mittal e M. K. Sharma, "A systematic review on sustainable E-waste management: challenges, circular economy practices, and a conceptual framework," *Management of Environmental Quality: An International Journal*, v. 35, n. 4, pp. 858–884, dez. de 2023, ISSN: 1477-7835. DOI: 10.1108/MEQ-05-2023-0139.

[7] J. P. d. T. Gomes e M. D. d. M. Innocentini, "Análise Socioambiental da Utilização de Internet das Coisas (IoT) na Gestão de Recursos Hídricos na Agricultura Familiar," em *Anais do VI Encontro de Desenvolvimento de Processos Agroindustriais (EDPA)*, Uniube, UFTM, IFTM, Uberaba, Brasil, 9 de dez. de 2022. endereço: https://www.researchgate.net/publication/373514301_ANALISE_SOCIOAMBIENTAL_DA_UTILIZACAO_DE_INTERNET_DAS_COISAS_IoT_NA_GESTAO_DE_RECURSOS_HIDRICOS_NA_AGRICULTURA_FAMILIAR.

[8] I. de Almeida, R. Czerniej, G. Galante e M. Oyamada, "Computing Continuum Implementation Using a FaaS Architecture with Repurposed TVBoxes," en, em *2025 XV Brazilian Symposium on Computing Systems Engineering (SBESC)*, Cascavel, Brasil: IEEE, 2025. DOI: 10.1109/SBESC60008.2025.11288815.

[9] C. R. M. Silva, J. P. N. dos Santos e F. A. C. M. Silva, "Empowering Hands-On Engineering Education with a Low-Cost IoT Hub Featuring Natural Language Support on Modified TV Boxes," en, em *2025 SBMO/IEEE MTT-S International Microwave and Optoelectronics Conference (IMOC)*, Natal, Brasil: IEEE, 2025, pp. 577–581. DOI: 10.1109/IMOC65414.2025.11365825.

[10] T. G. Bannwart, G. P. C. P. da Luz, G. M. Sato, L. F. G. Gonzalez e J. F. Borin, "TV Boxes as Support Servers for IoT Applications," en, em *Anais do I Workshop on Sustainable Computing and Technology Reuse (SCoRe 2025)*, Campinas, Brasil: Sociedade Brasileira de Computação (SBC), 26 de nov. de 2025, pp. 1–4. DOI: 10.5753/score.2025.17175.

[11] M. J. B. d. Sousa, "Design and Evaluation of a Method for Over-The-Air Firmware Updates for IoT Devices," en, Dissertação de Mestrado, Instituto de Computação, Universidade Estadual de Campinas, Campinas, Brasil, 2022. acesso em 6 de jul. de 2026. endereço: https://repositorio.unicamp.br/Acervo/Detalhe/1265519.

[12] G. P. C. P. da Luz, A. M. Aspilcueta Narvaez, T. G. Bannwart, G. M. Sato, L. F. G. Gonzalez e J. F. Borin, "Spot-Wise Smart Parking: An Edge-Enabled Architecture with YOLOv11 Towards a Digital Twin," en, *Journal of Internet Services and Applications*, v. 17, n. 1, pp. 110–120, 24 de abr. de 2026. DOI: 10.5753/jisa.2026.6664.