UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Thiago Haigerti Bertoldi

Gerência de Conectividade em um Gateway para Internet das Coisas

Florianópolis
2022

Thiago Haigerti Bertoldi

Gerência de Conectividade em um Gateway para Internet das Coisas

Relatório final da disciplina DAS5511 (Projeto de Fim de Curso) como Trabalho de Conclusão do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Catarina em Florianópolis.
Orientador: Prof. Carlos Barros Montez, Dr.
Supervisor: Sandro Alberton Kirchner, Eng.

Florianópolis
2022

Thiago Haigerti Bertoldi

Gerência de Conectividade em um Gateway para Internet das Coisas

Esta monografia foi julgada no contexto da disciplina DAS5511 (Projeto de Fim de Curso) e aprovada em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação

Florianópolis, 21 de março de 2022.

Prof. Hector Bessa Silveira, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Carlos Barros Montez, Dr.
Orientador
UFSC/CTC/DAS

Sandro Alberton Kirchner, Eng.
Supervisor
Khomp

Prof. Werner Kraus Junior, Dr.
Avaliador
UFSC/CTC/DAS

Prof. Eduardo Camponogara, Dr.
Presidente da Banca
UFSC/CTC/DAS

Este trabalho é dedicado a minha parceira, aos meus
colegas de classe e aos meus queridos pais.

AGRADECIMENTOS

Agradeço aos meus pais, pelo apoio incondicional durante os anos de graduação, e também a equipe de desenvolvimento da Khomp, que me ajudou a criar as bases da minha carreira profissional. Dos colegas da Khomp, cito diretamente meu gestor, Sandro Alberton Kirchner, e também os colegas Adriano Zaniboni de Pellegrin e Monique Aguiar Garcia. Trabalhar com todas essas pessoas incríveis é, sem dúvidas, muito inspirador. Por fim, estendo o meu agradecimento aos professores da UFSC, em especial ao professor Carlos Barros Montez, que me orientou na execução de dois projetos, ao professor Leandro Buss Becker, cujos direcionamentos me motivaram a concluir minha graduação, e também a professora Lúcia Helena Martins Pacheco, que sem dúvidas conduziu algumas das aulas mais proveitosas que tive na UFSC.

“Sim, eu consigo com uma pequena ajuda dos meus amigos”
(LENNON e MCCARTNEY, 1967, tradução nossa)

RESUMO

Long Range Wireless Area Network (LoRaWAN) é uma tecnologia emergente que permite comunicações sem fio a longa distância. A medida que esse tipo de rede se consolida, as empresas que lidam com esse padrão sentem necessidade de uma integração com outros protocolos de comunicação. Um exemplo concreto é a integração com os protocolos tradicionais da internet, como o HTTP. Redes de sensores sem fio (RSSF) já consolidadas, como a IEEE 802.15.4, são utilizadas para a comunicação e monitoramento de ”última milha” em sistemas de Internet das Coisas (IoT), enquanto o LoRaWAN cumpre o papel de comunicação de longo alcance. A Khomp se posiciona como uma empresa que desenvolve hardware e software para os seguimentos de telecomunicações, controle de acesso e IoT. Buscando parceiros integradores para comporem soluções IoT completas com seus produtos, um ponto central das soluções propostas que usam os produtos da Khomp é o gateway para Internet das Coisas ITG200, que faz a ponte entre as redes de sensores sem fio IEEE 802.15.4 e LoRaWAN com a internet. Para isso, o produto deve ser capaz de gerenciar suas interfaces de rede, sejam elas cabeadas ou baseadas em redes móveis, para que o produto possa rotear de forma adequada os dados da rede de sensores para o servidor dos parceiros integradores e vice-versa. Com a progressão das tecnologias de redes móveis, o gateway precisou se manter atualizado, comportando redes 4G com essa finalidade. Dessa forma, o objetivo do trabalho é descrever o projeto de software necessário para suportar as tecnologias de redes utilizadas pelo gateway, sejam elas as originais (3G) e as novas (4G). Para isso, se desenvolve um trabalho de software em cima da gerência de conectividade do gateway, permitindo assim o suporte às redes 4G e a correção de diversos outros problemas relacionados à conectividade.

Palavras-chave: Internet das Coisas (IoT). Software Embarcado. Redes de Computadores.

ABSTRACT

LoRaWAN is an emerging technology that enables wireless communications over long distances. As this type of network consolidates, companies that deal with this standard feel the need for integration with other communication protocols. A concrete example is the integration with traditional internet protocols, such as HTTP. Consolidated wireless sensor networks (WSNs), such as IEEE 802.15.4, are used for ”last mile” communication and monitoring in IoT systems, while LoRaWAN fulfills the role of long range communication. Khomp positions itself as a company that develops hardware and software for the segments of telecommunications, access control and IoT. Looking for integrator partners to compose complete IoT solutions with their products, a central point of the proposed solutions using Khomp products is the ITG200 Internet of Things gateway, which bridges the IEEE 802.15.4 and LoRaWAN wireless sensor networks with the internet. For this, the product must be able to manage its network interfaces, whether wired or based on mobile networks, so that the product can properly route data from the sensor network to the integrator partners' server and vice versa. With the progression of mobile network technologies, the gateway had to keep up to date, supporting 4G networks for this purpose. Thus, the objective of this work is to describe the software project necessary to support the network technologies used by the gateway, whether the original (3G) and the new (4G) technologies. For this, there is a software work to be developed on top of the gateway's connectivity management, for implementing support for 4G networks and correcting several other connectivity problems.

Keywords: Internet of Things (IoT). Embedded Software. Computer Networks.

LISTA DE FIGURAS

LISTA DE ABREVIATURAS E SIGLAS

API	Application Protocol Interface
CRUD	Create, Read, Update, Delete
FSF	Free Software Foundation
GPLv2	GNU General Public License versão 2
HTTP	HyperText Transfer Protocol
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
IoT	Internet das Coisas
IP	Internet Protocol
LoRaWAN	Long Range Wireless Area Network
LTE	Long Term Evolution
NTP	Network Time Protocol
POSIX	Portable Operating System-IX
PPP	Point to Point Protocol
TOML	Tom's Obvious, Minimal Language
UDS	Unix Domain Sockets
VPN	Virtual Private Network

SUMÁRIO

1 INTRODUÇÃO

IoT é um novo paradigma que combina aspectos de tecnologias como computação ubíqua, protocolos de internet, tecnologias de sensoriamento, tecnologias de comunicação e dispositivos embarcados, com a finalidade de criar um sistema onde o mundo real e digital se encontram e estão em uma interação simbiótica contínua, parafraseando (BORGIA, 2014). Graças as suas características (dispositivos independentes interligados), a IoT pode ser extensivamente utilizada para aplicações de telemetria remota. O potencial de aplicabilidade das soluções IoT é imenso, e a seguir são citados alguns cenários reais em que se pode fazer uso desse tipo de tecnologia.

- Soluções corporativas, monitorando conforto térmico e acústico, uso de energia elétrica, presença de funcionários entre outros;
- Soluções do agronegócio, para coleta automática de dados como temperatura e umidade do solo;
- Soluções da cadeia de frios, alimentação e hospitalar, verificando se insumos são armazenados de forma adequada;
- Na indústria, monitorando equipamentos críticos;
- Em cidades inteligentes, para coleta de dados de hidrômetros, fotocélulas e quaisquer outros dispositivos capazes de gerarem informações relevantes;

É perceptível que as soluções IoT podem se tornar bastante complexas, dado o número enorme de dispositivos interagindo (uma cidade conta com milhares de hidrômetros e fotocélulas, por exemplo). Essa complexidade toda pode ser dividida e classificada.

Assim, uma solução IoT completa conta com diversos elementos, em camadas bem definidas, sejam dispositivos de interação com o mundo real, sejam elementos de borda, sejam elementos de nuvem. Detalharemos a estrutura geral de uma solução IoT e suas camadas e elementos a seguir. No final da introdução, há um breve resumo da organização dos capítulos da monografia.

1.1 ESTRUTURA DE UMA SOLUÇÃO IOT

O termo Solução IoT descreve todos os componentes de software e hardware IoT utilizados para a geração de valor para os usuários. Nessa seção iremos definir em detalhes as camadas de uma solução IoT, apresentaremos uma arquitetura simplificada (no contexto da utilização do produto desenvolvido) e os elementos dessa topologia de rede.

1.1.1 Camadas

Segundo (KHAN *et al.*, 2012), uma solução IoT pode ser dividida em cinco camadas, ordenadas na listagem a seguir de forma que as camadas mais próximas do mundo real e tangível são citadas primeiro.

- A Camada de Percepção;
- A Camada de Rede;
- A Camada de Middleware;
- A Camada de Aplicação;
- A Camada de Negócio;

A camada de percepção é onde estão os sensores e atuadores da solução IoT. Eles interagem entre si através da camada de rede, que engloba também a comunicação da camada de percepção com a camada de middleware. Na camada de middleware, é feito o roteamente das informações para programas e serviços da camada de aplicação. Nesses programas e serviços há a camada de negócio, onde se gera o verdadeiro valor da solução para os usuários finais (relatórios, alarmes, painéis de controle, etc).

O presente trabalho trata, então, de um elemento específico da camada de rede: o Gateway IoT. Esse elemento e os demais elementos necessários para a compreensão de como funciona uma solução IoT são descritos a seguir.

1.1.2 Arquitetura e Elementos

Uma solução IoT genérica conta com os elementos da Figura 1, onde a rede de sensores consiste numa série de dispositivos sem fio se comunicando via radiofrequência (camada de percepção), o Gateway é o ponto central da solução (camada de rede), fazendo a ponte entre a rede sem fio e a internet, onde as demais camadas são omitidas (estão representadas dentro da estrutura de nuvem).

Figura 1 – Arquitetura simplificada

Desses três elementos, o que terá maior destaque nesse projeto é o Gateway IoT. Para definir o que será feito no trabalho, é necessário primeiramente conceitar a Gerência de Conectividade no contexto do Gateway IoT.

1.2 GERÊNCIA DE CONECTIVIDADE

Como é possível de se observar pela topologia da solução, é essencial que as informações da rede de sensores possam ser enviadas para a internet e vice-versa. Nesse sentido, a gerência de conectividade se transforma num ponto crítico da solução IoT. Isso porque, em geral, um Gateway para Internet das Coisas pode contar com inúmeras interfaces de rede, cabeadas ou baseadas em tecnologias móveis como o 3G e 4G. Para garantir o fluxo de informações entre a rede de sensores sem fio e a internet, o Gateway deve rotear as mensagens pelas interfaces de rede adequadas, mantendo um controle da qualidade de conexão para cada interface e implementando uma política de uso para elas. Esse é o tema central deste trabalho: a gerência de conectividade de um Gateway para Internet das Coisas.

1.3 ESTRUTURA DO TRABALHO

O produto desenvolvido neste trabalho é uma nova versão de software para um gateway IoT chamado ITG 200, um produto da empresa Khomp, utilizado nos diversos contextos já citados (soluções hospitalares, industriais, logísticas, agronegócio, cidades inteligentes, etc). A justificativa inicial para que houvesse um trabalho de sofware é a modernização de hardware, que agora inclui um módulo modem 4G. Isso demandou um trabalho de integração extenso. Como a mudança de software não é trivial, encaixamos a resolução de vários outros problemas de conectividade na mesma versão, de forma que a gerência de conectividade do Gateway foi totalmente remodelada, com o intuito de se remover a gerência de conectividade antiga, que apresentava problemas.

No Capítulo 2 é feita uma contextualização do produto, estabelecendo quais são as bases de hardware e software para o desenvolvimento, com a finalidade de apresentar a motivação (problemas a serem resolvidos), objetivos e proposta de solução.

No Capítulo 3 a empresa Khomp é apresentada, com sua história, política de qualidade e estrutura gerencial. Nesse momento, são apresentadas também as ferramentas internas da equipe de desenvolvimento.

No Capítulo 4, são explicadas as tecnologias envolvidas na composição da solução, em todas as camadas onde houve desenvolvimento.

No Capítulo 5 são apresentados os requisitos gerais, funcionais e não-funcionais.

No Capítulo 6 é apresentado o projeto de software, com um detalhamento das escolhas feitas e detalhes de implementação.

No Capítulo 7 ocorre a discussão acerca dos resultados esperados pelo trabalho, as métricas de avaliação e os resultados obtidos.

No Capítulo 8 são propostas possíveis melhorias sobre a gerência de conectividade desenvolvida e novas funcionalidades que poderão ser implementadas graças a ela.

No último capítulo, é apresentada a conclusão do trabalho, com uma retrospectiva dos principais pontos de desenvolvimento.

2 O PRODUTO

O produto desenvolvido neste trabalho é o ITG 200, um gateway para telemetria IoT projetado e desenvolvido pela Khomp, do hardware até o software. Esse gateway suporta dispositivos com as tecnologias de comunicação a rádio LoRaWAN ou IEEE 802.15.4 (Zigbee), de acordo com o módulo de rádio conectado no produto.

O gateway ITG 200 (representado na Figura 2 em seu modelo Indoor) conta também com entradas 1-wire para comunicação com sensores de temperatura do tipo DS18B20, entradas para contatos secos e duas portas ethernet para conexões cabeadas. Ele conta também com um slot para modem 3G (com dois SIM cards), e a partir do resultado deste trabalho, esse mesmo slot pode ser utilizado para a conexão de modems 4G (também com dois SIM cards).

Figura 2 – ITG 200 indoor

Fonte: Site Khomp.

2.1 FUNCIONALIDADES DO PRODUTO

Um gateway é, por definição, o elemento de rede responsável por fazer uma ponte entre redes diferentes. No caso do ITG 200, essas redes são a rede de sensores sem fio e a internet. A nível de gerência de dispositivos, o ITG 200 é capaz de administrar redes LoRaWAN e IEEE 802.15.4. A nível de gerência de rede, são várias funcionalidades:

- Configuração de rede com DHCP ou IP fixo, máscara de rede e gateway padrão;
- Configuração de lista de DNS;
- Configuração de credenciais para acesso a internet via rede móvel;

- Sincronização de data e hora via NTP;
- Envio de dados de sensores via HTTP;
- Envio de dados de sensores via MQTT;
- Atualização remota via nuvem de atualização;
- Acesso remoto com OpenVPN

2.2 HARDWARE

O ITG 200 conta conta com um processador ARMv7 de 700MHz, single core. Ele contém 128MB de armazenamento em memória não volátil do tipo nand e 256MB de memória volátil. Seu hardware comporta uma distribuição de Linux Embarcado proprietária da Khomp (chamada Kharma), similar em funcionalidade ao Yocto e Buildroot. Quanto aos seus módulos de rede, no caso da Ethernet, são duas interfaces RJ45. Já o módulo 3G tem as seguintes características:

- Bandas de operação 3G: B5, B8, B2, B1, B4
- Bandas de frequência: 800/850, 900, AWS1700, 1900, 2100 MHz
- Transmissão HSPA: 21/5,7 Mbps (download/upload)
- Transmissão WCDMA: 384/384 Kbps (download/upload)
- Transmissão EDGE: 296/236 Kbps (download/upload)
- Transmissão GPRS: 107/85.6 Kbps (download/upload)
- Suporta 2 SIM cards do padrão Mini SIM (2FF)

2.3 ARQUITETURA ORIGINAL

A arquitetura de software original do gateway está representada no diagrama da Figura 3. Nessa imagem, estão representados alguns elementos de software importantes o trabalho. Os principais são o firmware do Módulo 3G e sua interação com a aplicação Python chamada de ITG Core, onde está a lógica de negócios do produto. Essa aplicação configura e gerencia as interfaces de rede cabeadas e via modem através de abstrações do Linux e de comandos AT via porta serial.

Os demais elementos representados lidam com os módulos de rede sem fio IEEE 802.15.4 e LoRaWAN, já que o Gateway pode administrar esses dois tipos de redes de sensores sem fio.

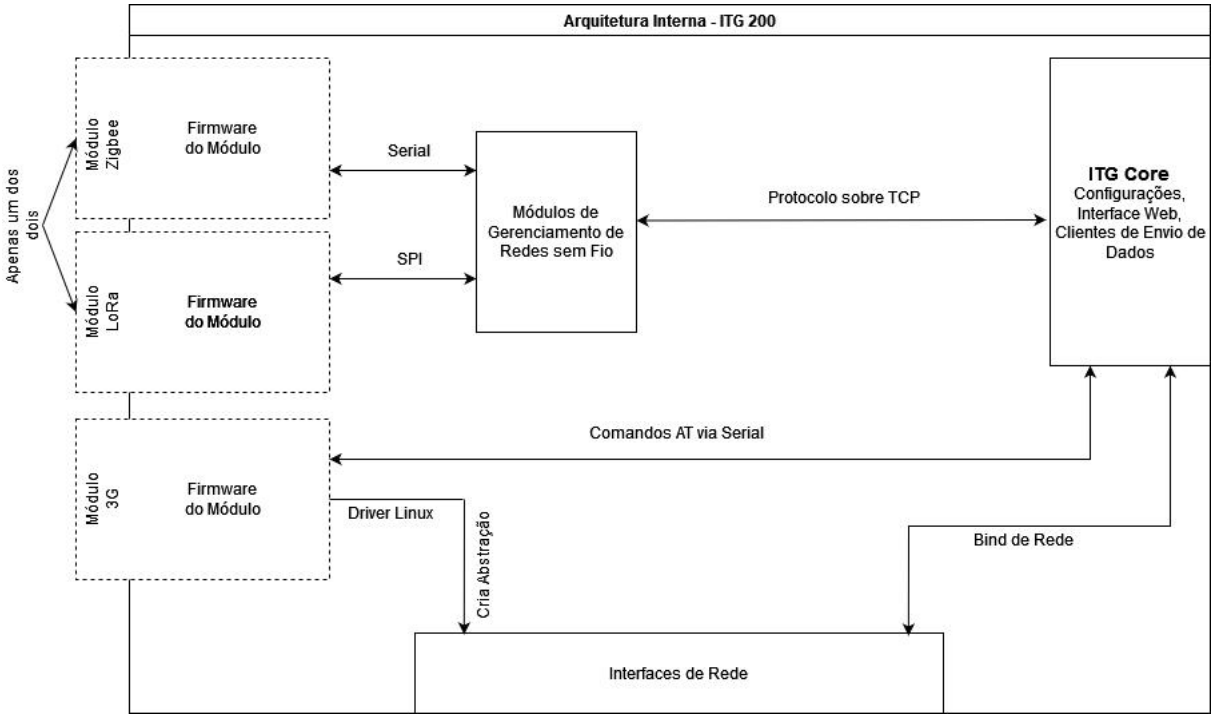


Figura 3 – Arquitetura de software original ITG 200

2.4 INICIALIZAÇÃO DO SOFTWARE ORIGINAL

Discutiremos aqui o ITG Core, aplicação Python que concentra toda a lógica de negócios do Gateway. O ponto de entrada da aplicação é o módulo que carrega as configurações. A partir das configurações, o módulo coordenador interno inicializa os serviços configurados. Os serviços disponibilizam funcionalidades como a atualização remota do ITG, o acesso via VPN e a sincronização de data e hora via NTP. Além disso, as interfaces de rede detectadas são configuradas e habilitadas para o envio de dados. O coordenador também cuida das entradas e saídas básicas, como o display, botões, leds e o buzzer (Figura 4.

Outro módulo que interage com o controlador é o de agenciamento de redes sem fio. É a partir dele que o controlador identifica a rede sem fio associada ao Gateway. O gerenciamento de redes sem fio depende da troca de mensagens internas entre módulos sem fio e o restante da aplicação. Os dados relevantes são salvos no Banco de Dados através do ORM, que também é consultado pelos clientes externos responsáveis por fazerem os envios de dados para a nuvem, de acordo com os protocolos de rede habilitados. Dados da rede sem fio que não demandam persistência (e por isso não estão associados ao Banco de Dados) transitam pelo coordenador e pelo gerenciamento de clientes externos diretamente.

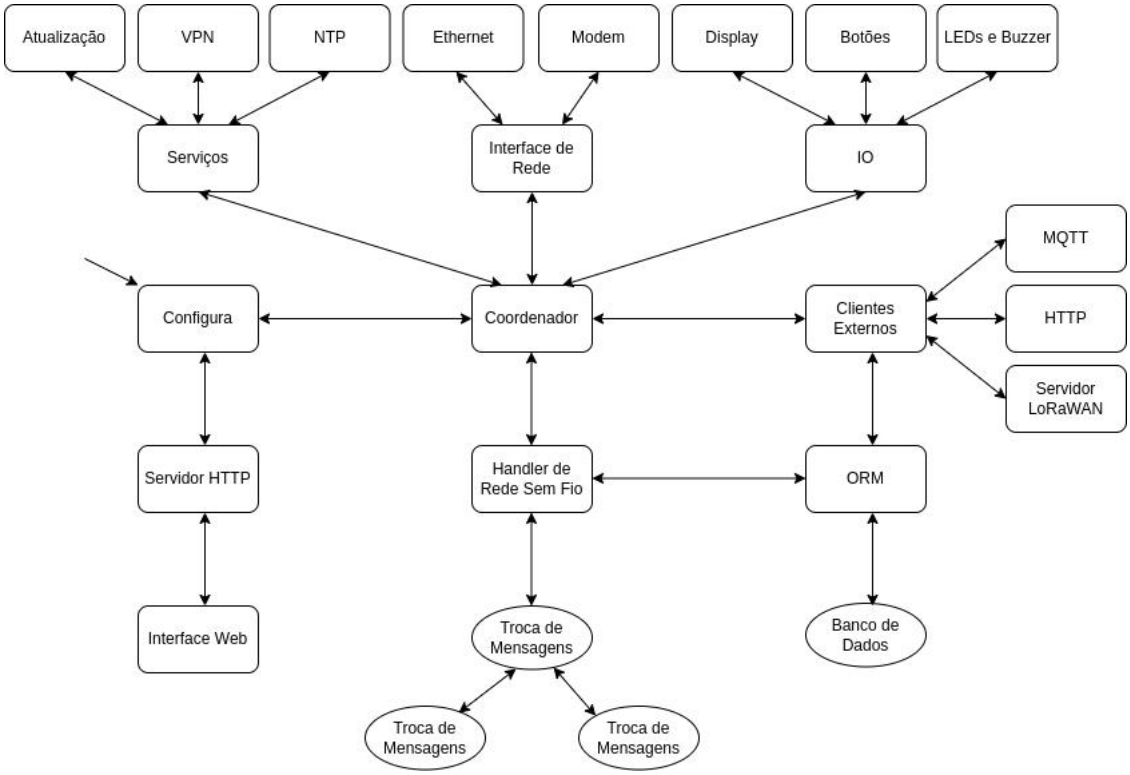


Figura 4 – Inicialização de software original ITG 200

2.5 MOTIVAÇÃO

Com a evolução das redes móveis, surgiu a demanda, por parte de parceiros da Khomp, de que a empresa disponibilizasse opções de modens 4G. Assim, a gerência da Khomp especificou que, a partir do hardware existente e de um novo módulo 4G, se deseja criar uma nova versão do ITG 200 com suporte a todas as funcionalidades antigas com o acréscimo da integração do modem 4G.

Com base nesse objetivo, a equipe de Hardware do ITG 200 montou um módulo 4G baseado no modem Quectel 4G (que já era empregado em outros produtos da Khomp, de forma que a integração desse modem em particular seria mais fácil, e os custos associados a estoque seriam menores).

As características do modem Quectel 4G são as seguintes:

- Bandas de operação LTE FDD: B1, B2, B3, B4, B5, B7, B8, B12, B13, B18, B19, B20, B25, B26, B28
- Bandas de operação LTE TDD: 838, 839, 840, 841
- Bandas de operação UMTS: B1, B2, B4, B5, B6, B8, B19
- Bandas de operação GSM: B2, B3, B5, B8
- Transmissão LTE: 150/50 Mbps (download/upload)

- Transmissão UMTS: 384/384 Kbps (download/upload)
- Transmissão GSM: 296/236 Kbps (download/upload)
- Suporta 2 SIM cards do padrão Micro SIM (3FF)

O novo modem não é suportado pelos drivers do ITG 200, de forma que é necessário realizar alterações de software no Linux Embarcado e nas aplicações internas do Gateway. Além da inclusão do novo modem 4G, se deseja aproveitar as mudanças na gerência de conectividade para corrigir problemas existentes na atual gerência de conectividade do Gateway. Os problemas são os seguintes:

- Utilizar uma rede móvel para envio de dados remove o acesso ao Gateway via interface cabeada;
- Em casos de problema de conectividade, a árvore de decisões do Gateway pode levar a reboots (que são indesejados, pois desligado o Gateway não pode gerênciar as redes sem fio);
- Interfaces de rede não podem estar na mesma faixa de IPv4, caso contrário o Gateway apresenta comportamento anômalo;
- Os testes de conectividade não são configuráveis, e o resultado fica "escondido"em meio aos outros logs da aplicação;
- A troca de interface de rede em uso para envio de dados é muito lenta no pior caso (cerca de 15 minutos).

Além da resolução desses problemas, se deseja permitir que o usuário tenha acesso a mais métricas de uso das interfaces de rede e informações sobre modem e SIM card, para ser capaz de diagnosticar problemas mais rapidamente e entender melhor o comportamento do Gateway no que tange a gerência de conectividade.

2.6 PROPOSTA DE SOLUÇÃO

Com base nisso, é proposta uma nova arquitetura de software para o Gateway ITG 200, de forma a dar suporte ao modem 4G e sanar todos os problemas citados na seção de motivação. Essa nova arquitetura tem o compromisso de ser compatível com os produtos já distribuídos, ao mesmo tempo que tem a liberdade para utilizar ferramentas avançadas do Linux, como o suporte às múltiplas tabelas de roteamento IP.

Como o modem 4G já era empregado por outras equipes da Khomp, é natural de que a arquitetura proposta fosse baseada nos demais produtos. Detalhes sobre a nova arquitetura, junto das justificativas para as escolhas feitas, estão em capítulos posteriores (Projeto de Software).

2.7 METOLOGIAS APLICADAS

A fim de se implementar a proposta de solução, foram executados os seguintes passos:

- Concepção da Arquitetura do Sistema;
- escolha das tecnologias a serem empregadas;
- desenvolvimento de software empregando metodologias ágeis;
- validação por parte da equipe de QA do resultado do desenvolvimento.

3 A EMPRESA E OS PROCESSOS

Nesta seção teremos uma breve introdução à empresa Khomp, apresentando sua história, política de qualidade, visão estratégica, e ferramental sugerido para as equipes internas.

3.1 HISTÓRIA DA EMPRESA

A Khomp é uma empresa de tecnologia brasileira com matriz em Florianópolis, Polo de Inovação em Santa Catarina. Ela foi fundada em 1996 com o esforço principal voltado para a área de centrais de comutação telefônica pública e privada. Seu objetivo era proporcionar maior longevidade aos equipamentos existentes através da incorporação de novas facilidades, tornando-os mais competitivos e tecnologicamente atualizados. Com uma equipe técnica treinada e especializada em processamento digital de sinais, essa empresa passou a atuar no mercado desenvolvendo projetos personalizados em hardware e software, que resultaram em produtos de alto desempenho.

A partir de 2001, aplicando o conhecimento e a experiência adquiridos desde sua fundação, a Khomp passou a desenvolver produtos voltados ao mercado CTI (Computer Telephony Integration). A sua linha de placas PCI em conjunto com os sistemas desenvolvidos por seus clientes (integradores), se destinam as mais diversas aplicações onde a integração computador e telefonia é empregada.

Em 2007 a Khomp lançou-se em um novo mercado de aplicações que passaram a usar software livre, como o Asterisk® e o FreeSWITCH™, o mercado Soft PBX. A experiência de sua equipe no desenvolvimento de soluções permitiu ingressar nesse mercado e lançar em 2008 as placas SPX, para aplicações open source.

Sempre buscando inovação, em 2009 a Khomp passa a integrar media gateways com a mais alta densidade do mercado em seu portfólio, os chamados Kmedias e, posteriormente em 2012, amplia essa família de produtos com os KMG, media gateways de baixa densidade. O ano de 2009 também trouxe o primeiro Escritório Regional da Khomp na cidade de Buenos Aires, Argentina, marcando o início de sua trajetória no exterior.

Mais uma vez inovadora, em 2011, a Khomp lança uma nova linha de produtos chamada External Board Series. Os EBS são módulos compactos de 1U e meio rack para todas as interfaces de telefonia trabalhando fora do servidor, o que permite fácil ampliação de cenários, criação de sistemas redundantes, entre outras facilidades, mantendo o padrão e a qualidade da família de placas Khomp. Os EBS da Khomp trouxeram um novo conceito de hardware para o mercado de telecom.

Em paralelo ao lançamento de sua linha de gateways de baixa densidade, a Khomp lança o EBS Server, um appliance que possibilita ao integrador desenvolver um

produto único com aplicativo, processamento e conexão de interfaces. Além destes lançamentos, outras duas novidades em seu portfólio: uma família de intercomunicadores de acesso via rede IP e também uma linha de telefones IP.

Ampliando sua atuação no mercado internacional, em 2013 a Khomp inaugura outro Escritório Regional, desta vez na Cidade do México, capital mexicana. A linha Appliance ganha novos membros, como o EBS Server GSM para soluções celular e o EBS Server Light, uma versão somente E1 e compacta de seu predecessor.

Em busca de novidades para as soluções de seus clientes, em 2014, a Khomp ampliou seu portfólio de dispositivos IP com a linha Endpoints SIP, agregando valor às soluções de telefonia destes através da integração com sistemas de segurança e controle de acesso. Neste mesmo ano, essa empresa lançou seu programa de workshops gratuitos, os Workoffees Khomp. Os encontros aconteceram em diversas cidades do Brasil e possibilitam ampliar o conhecimento técnico de seus parceiros e clientes.

O começo de 2018 foi marcado pelo lançamento do Khomp Academy, um laboratório de treinamentos presenciais focado na capacitação técnica de parceiros. A empresa percebe nessa maior aproximação com as pessoas que sem mentes criativas, a tecnologia não funciona, e que tecnologia desconectada da sociedade é puramente estéril. Assim, nasce um novo propósito de negócio para a Khomp: Enabling Technology, ou Viabilizar Tecnologia, em tradução livre.

Também em 2018, a Khomp inicia a liberação de sua linha de produtos IoT, produtos pensados para o mercado nacional e competitivos tecnologicamente e em preço, com desenvolvimento totalmente nacional. Nesse setor, são desenvolvidos os Gateways e Endpoints IEEE 802.15.4 e LoRa, suportando diferentes cenários de aplicações com essas tecnologias.

Em 2021, visando ganhar ainda mais musculatura para realizar objetivos e fortalecer estratégias nas áreas de telecomunicações e Internet das Coisas, o Grupo Intelbras se torna sócio da Khomp. O interesse na concretização dessa parceria surgiu a partir da relação próxima dos clientes de ambas empresas, as quais possuem filosofia e princípios semelhantes.

3.2 MISSÃO, VISÃO E POLÍTICA DE QUALIDADE DA EMPRESA

A Khomp define como sua missão obter um relacionamento que orgulha parceiros e colaboradores com resultados duradouros, atitude criativa e inovadora em produtos e serviços com tecnologia de comunicação e informação.

Como visão, a Khomp deseja ser reconhecida como referência por seus resultados, inovação, ambiente único e projeção global. Como política de qualidade, a Khomp busca a satisfação de seus clientes, colaboradores, fornecedores e acionistas mediante:

- Flexibilidade e comprometimento na parceria com seus clientes e fornecedores
- Desenvolvimento de produtos de qualidade evoluindo continuamente seus processos
- Capacitação e motivação de sua equipe
- Transparência e rentabilidade

3.3 SISTEMA DE GESTÃO DE QUALIDADE

A Khomp conta com um Sistema de Gestão de Qualidade bem definido e acessível para todos os funcionários, estagiários e menores aprendizes através de um site interno. Pela sua padronização de processos, a Khomp conta com Certificaçãol ISO 9001, que atesta a qualidade do trabalho da empresa e concede benefícios fiscais.

A ISO 9001 é um sistema de gestão com o intuito de garantir a otimização de processos, maior agilidade no desenvolvimento de produtos e produção mais ágil a fim de satisfazer os clientes e alcançar o sucesso sustentado. Dos processos definidos, o mais pertinente para o presente relatório é o ciclo de vida de um produto na empresa. Em termos gerais, um produto passa pelos seguintes estágios:

- Concepção, onde são definidos os alvos de preço e funcionalidades;
- Prototipação, onde o produto é esboçado e assume sua primeira forma;
- Teste de Campo, onde o produto é colocado em situações reais e é avaliado;
- Liberação Comercial, quando o produto desenvolvido é anunciado para outros setores da empresa.

Cada setor da empresa irá contribuir de alguma forma para o sucesso do produto. A equipe de Documentação irá assegurar que o produto está disponível e bem descrito; A equipe de Engenharia de Produto irá arquitetar os detalhes da produção dos lotes do produto; A equipe de Marketing será responsável por criar o material de divulgação do produto; A equipe comercial irá prospectar novos negócios e criar ações para vender o produto; A equipe de Suporte irá cuidar das dúvidas de clientes relativas ao produto. Todos esses processos relativos ao produto seguem especificações bem claras e definidas, onde cada setor se vale do SGQ para garantir suas entregas. Detalhar os processos de outros setores foge do escopo deste relatório de projeto de fim de curso.

3.4 MAPA DE EQUIPES

Dentro da empresa, são vários os departamentos. Especificamenente no departamento de Pesquisa e Desenvolvimento, há uma equipe de IoT. Essa equipe é

divida primeiramente entre os responsáveis pelo Hardware dos produtos, os responsáveis pelo Software dos Produtos e os responsáveis pelo Controle de Qualidade dos Produtos. Dentre os responsáveis pelo Software dos Produtos, há uma divisão mais flexível entre os responsáveis por cada produto. Dessas equipes, a maior é a equipe de Software do ITG 200, justamente por ele ser o produto central das soluções de IoT da Khomp. Há uma linha de IoT que não depende do ITG 200, os chamados ITS, dispositivos com conexão própria a internet. Esses dispositivos não serão citados no relatório já que não tem relação com o projeto realizado.

3.5 FERRAMENTAS DA EQUIPE

Para um membro de uma equipe de desenvolvimento de software, há uma série de ferramentas específicas para garantir o fluxo de trabalho. Assim como os gestores utilizam ferramentas para acompanhar os projetos, desenvolvedores de software utilizam ferramentas para garantir o fluxo de trabalho com os códigos das aplicações utilizadas no produto. São três as principais ferramentas organizacionais: o gerenciador de projetos, o sistema de controle de versão e a ferramenta de comunicação oficial da equipe.

3.5.1 Comunicação Interna

A Khomp utiliza como ferramenta de comunicação interna dos times de desenvolvimento o Mattermost e o Google Chat, que são plataformas que permitem a troca de mensagens entre usuários e grupos, além de permitir a criação de canais de áudio para conferências e o compartilhamento de telas para apresentações.

Outra ferramenta de comunicação utilizada é a rede social corporativa Bizu, derivada da SocialBase. Ela se resume a um feed e um conjunto de grupos, no qual diversas equipes e usuários podem compartilhar conteúdos a respeito de novos produtos, oportunidades e miscelânceas.

3.5.2 Gerência de Projetos

Como Gerenciador de Projetos de Desenvolvimento de Software, a Khomp utiliza o Redmine. Dentro da plataforma, novas tarefas a serem realizadas são descritas em um formato de ”Histórias de Usuário”. Histórias de Usuário definem três elementos centrais na descrição de uma tarefa: ”O quê?”, ”Por quê?” e ”Como?”. Um detalhamento sobre o funcionamento de histórias de usuário está escrito a seguir.

Além da descrição no formato de histórias de usuário, a tarefa contém campos indicando:

- O produto associado;

- A versão em que a tarefa foi finalizada;
- O status da tarefa (em andamento, finalizada, em validação);
- O tipo da tarefa (melhoria, nova funcionalidade, correção de problema);
- Comentários de outros usuários da ferramenta a respeito da tarefa.

3.5.2.1 Histórias de Usuário

Uma história de usuário permite que a tarefa seja descrita em termos do que gera valor para o cliente que fará proveito da nova funcionalidade/correção/otimização. No campo ”O quê?” é descrito o propósito da tarefa. Por exemplo: ”Adicionar representação em termos percentuais para o nível de bateria dos dispositivos registrados”. Dessa forma, a intenção da tarefa é clara para quem irá escrever o código que implementa a funcionalidade. No campo ”Por quê?”, se assume a perspectiva de quem irá utilizar o recurso pronto para descrever como a tarefa gera valor para o produto. Para o mesmo exemplo, o campo poderia ser preenchido com ”Como um usuário, desejo saber de forma simples e intuitiva se o meu dispositivo está em condições de operar normalmente”. No último campo da história de usuário, é definido o ”Como?”. Para essa questão, é dado um indicativo de como proceder para gerar o valor para o usuário. No mesmo exemplo, uma resposta para esse campo seria ”Criando um método na classe XXXXX que retorna o valor percentual da bateria e usando essa informação na exibição da interface gráfica, no lugar do valor bruto”.

Histórias de usuários também são úteis como base para a descrição de testes para o software. Detalhes em capítulos posteriores.

3.5.3 Controle de Versão

Todo o setor de desenvolvimento da Khomp utiliza o Git para fazer o controle de versão do código. O Git é um programa capaz de controlar o fluxo de trabalho, separando as etapas de desenvolvimento entre:

- Criação de ambientes de trabalho;
- Escrita do código;
- Marcação de mudanças prontas para serem realizadas;
- Aplicação das mudanças de código no seu ambiente de trabalho.
- Aplicação das mudanças do seu ambiente de trabalho na base do código utilizada no produto.

A expressão utilizada para definir um ambiente de trabalho novo a partir de algum ponto é "branch", galho em tradução livre. O que denota a aplicação de uma mudança de código é um "commit", e o que define a aplicação das mudanças de um ambiente de trabalho em outro é um "merge".

A base de todo código é chamada de "master", como em mestre, por ser a fonte de verdade, ou seja, o local para qual o código validado retorna.

Separar o desenvolvimento em ambientes de trabalho facilita o trabalho em equipe, enquanto as etapas para criar uma alteração permanente abrem espaço para momentos de revisão de código por parte de colegas, facilitam a reversão de mudanças, estimulam a aplicação de mudanças de forma gradual, além de outros benefícios atrelados ao próprio versionamento.

A plataforma utilizada para conter todos os repositórios do Git utilizados no fluxo normal de desenvolvimento é o Gitlab - Edição da Comunidade.

O Gitlab disponibiliza uma interface gráfica disponível via internet, na qual é possível visualizar os repositórios, mudanças, usuários e projetos de forma simples.

O uso do Git permite diversos fluxos de trabalho, que parametrizam como as mudanças na base de código serão efetuadas. O fluxo de trabalho utilizado em todos os projetos da equipe de IoT é o chamado Fluxo do Gitlab.

3.5.3.1 Fluxo do Gitlab

São alguns conceitos utilizados no Fluxo do Gitlab. Os mais importantes são a compressão de commits, os ”Features branchs” e a política de ”Upstream First”. Quando é necessário gerar uma versão nova de software nesse fluxo, a tag da versão origina também um novo ambiente de trabalho. Explicando os conceitos:

A compressão de commits indica que, ao aplicar as mudanças de um ambiente de trabalho em outro, em termos de versionamento, não importa quantas mudanças foram feitas no ambiente de trabalho original, o processo de aplicar as mudanças irá contar como apenas uma mudança no ambiente de trabalho que está recebendo o código novo.

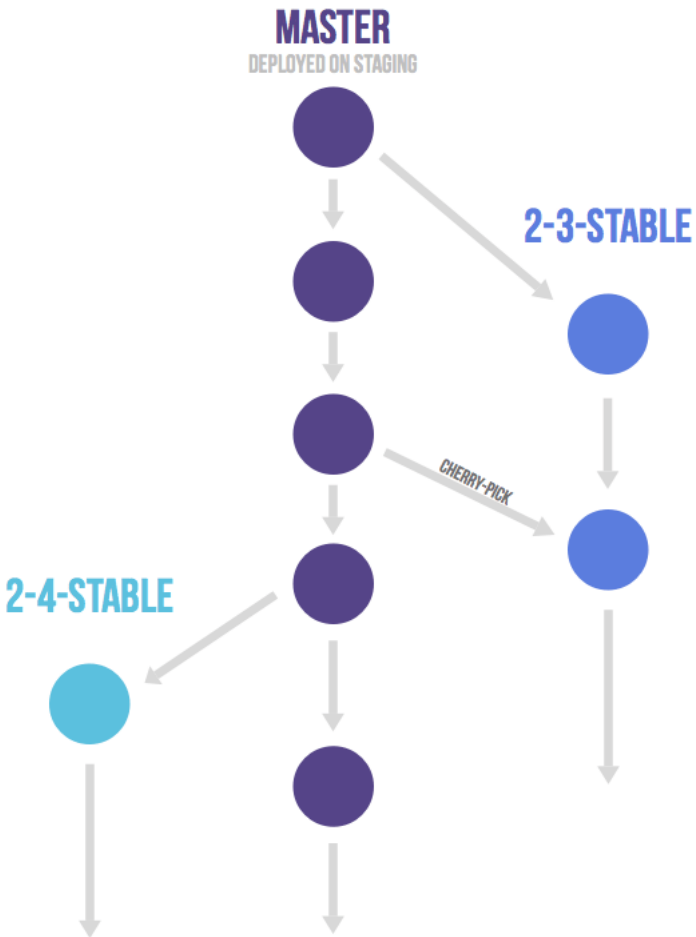
O conceito de ”Feature Branch”, ou ”galho de funcionalidade” consiste na ideia de que cada ambiente de trabalho gerado a partir do código original cumprirá apenas uma função, realizando pontualmente apenas uma tarefa. Isso faz sentido com o fluxo de trabalho do Redmine, porque cada ambiente de trabalho é mapeado diretamente a uma tarefa.

”Upstream first” tem a ver com o fato de que qualquer correção é feita em um branch que é combinado primeiramente ao master do projeto. Depois disso, caso a correção deva ser aplicada em outro ambiente de trabalho, a correção recebe um cherry-pick para o ambiente de trabalho necessitando da correção. Um merge não funcionaria porque carregaria o histórico de mudanças consigo, enquanto o cherry-pick

carrega apenas a mudança de um commit, e a ideia é de ter apenas a correção no outro ambiente de trabalho. Esse fluxo também assegura que toda correção sempre estará disponível em versões novas.

Aplicando todas as ideias desse fluxo (Figura 5), é possível ter segurança nas entregas das versões dos softwares utilizados no produto.

Figura 5 – Fluxo do Gitlab



Fonte: Site do Gitlab.

4 TECNOLOGIAS E DESAFIOS

4.1 LINGUAGEM GO

A linguagem de programação Go (também chamada de Golang) é uma criação dos desenvolvedores de software Robert Griesemer, Rob Pike e Ken Thompson, como uma linguagem voltada para resolver os problemas que os programadores da Google enfrentavam diariamente. Foi estipulado que seria necessário desenvolver uma linguagem com tempo de compilação baixo, gerência automática de memória e suporte nativo a operações de concorrência. Assim, o propósito do Go é ser uma linguagem de propósito geral, projetada com a programação de sistemas em mente, com uma tipagem forte, garbage collector e concorrência nativa, segundo a própria especificação da própria linguagem (GOOGLE, 2021).

Essas mesmas características de desempenho se provaram muito úteis para o segmento de Internet das Coisas, tendo em vista de que existem vários exemplos de projetos open-source baseados em Go, como é o caso das plataformas de IoT Grafana e Chirpstack. O Go é utilizado em sistemas embarcados e em serviços distribuídos de nuvem, atestando novamente a versatilidade da linguagem.

Assim, iremos explorar aspectos do suporte a concorrência, gerência de memória e compilação do Go.

4.1.1 Concorrência

O suporte a concorrência do Go, ainda segundo sua especificação, é dado de três formas: com channels, o pacote de sincronização e operações atômicas. Há uma recomendação que é consenso na comunidade de desenvolvedores, de que se for possível, usar preferencialmente channels (que são robustos e facilmente administrados por qualquer desenvolvedor com familiaridade com a linguagem), e, apenas se não for possível resolver o problema de concorrência com channels, empregar o pacote de sincronização ou as operações atômicas.

As threads do Go são geridas não pelo escalonador do sistema operacional, e sim pelo ambiente de runtime do Go. Assim, cada thread acrescenta um overhead de 4kb ao programa, de forma que em sistemas computacionais modernos, é possível de existirem milhares de threads simultâneas na aplicação. A esse conceito se dá o nome de ”Green Threads”, e a implementação de ”Green Threads” no Go é chamada de ”Go-Rotinas”.

Para sinalizar para o runtime do Go de que um método ou função deve ser executado numa Go Rotina, basta acrescentar o modificador ”go” na chamada da função. Assim, qualquer função pode se tornar uma função assíncrona, de acordo com a forma como é invocada. Para uma Go-Rotina trocar mensagens com outras

Go-Rotinas de forma segura, a linguagem oferece suporte ao elemento ”channel”, canais em tradução livre, que são, em síntese, estruturas com mutexes internos. Seu funcionamento está detalhado a seguir.

4.1.1.1 Channels

Essas estruturas do Go servem como mecanismo para troca de mensagens entre diferentes Go-Rotinas. Na prática, funciona como um buffer de mensagens, em que podem ser postadas mensagens e também de onde se podem consumir mensagens. A outra operação suportada pelos channels é a de ”close”, que fecha o canal e não permite mais comunicação. Postar um valor num canal fechado é um erro em runtime. Receber uma informação de um canal fechado simplesmente retornará um objeto vazio do tipo definido para o channel. A notação para inicializar um channel no Go é a seguinte:

```
canal := make(chan type)
```

Esse exemplo está criando um canal chamado ”canal” e com tipo ”type”. O tipo pode ser qualquer tipo nativo da lingagem (string, bool, int, int64 etc) ou qualquer tipo definido por usuários na forma de estruturas de dados. Na inicializaçõa do canal, também é possível definir um tamanho máximo de buffer, de forma que tentar postar uma mensagem quando o buffer estiver cheio se torna uma operação bloqueante.

Para enviar uma mensagem para um canal, a notação é:

```
canal <- mensagemIn
```

Para receber uma mensagem de um canal, basta usar a seguinte notação:

```
mensagemOut <- canal
```

Para fechar um canal, a notação utilizada é:

```
close(mensagemOut)
```

Essas quatro operações, combinadas, permitem uma comunicação assíncrona entre Go-Rotinas que é segura e determinística.

Os canais são então utilizados como parâmetros nas chamadas de função, de forma que sua integração com o restante da linguagem não demanda nenhuma notação específica.

Esse mecanismo, combinado com a estrutura ”select” do Go, é muito poderosa para aplicações concorrentes. O ”select” é uma estrutura similar a um switch case, no entanto, a operação a ser executada é sempre a primeira operação que não estiver bloqueante. Dessa forma, a implementação de timeouts é muito simples.

Em exemplos posteriores, do projeto de código, poderemos ver todos esses mecanismos em ação.

4.1.1.2 Pacote de Sincronização

O Go conta com um módulo específico para sincronização, onde estão as implementações básicas de algumas estruturas de sincronização, como os mutexes.

Para utilizar um mutex com Go, basta importar o pacote de sincronização "sync". Além do mutex, o pacote implementa diversas outras estruturas, como por exemplo sinais de sistema e etc. Assim, a recomendação dos profissionais que lidam com a linguagem é de que seja utilizado um mutex para proteger zonas críticas de código, quando o código em questão não puder ser implementado por meio de ”channels”.

Os mutexes são uma solução rápida para problemas de arquitetura de software, como poderemos ver detalhadamente no capítulo de implementação.

4.1.1.3 Operações Atômicas

Outra forma de ter operações seguras no Go é utilizando o pacote "atomic". Operações atômicas são operações que já são previstas pela linguagem assembly de algumas arquiteturas. Em geral, no contexto dos sistemas embarcados, operações atômicas são mais custosas (na métrica de tempo de execução).

Operações atômicas são suportadas apenas em tipos numéricos e inteiros, então seu uso é bem restrito para alguns poucos cenários, que não serão abordados no presente documento.

4.1.2 Gerência de memória

O Go conta com um ”garbage collector” para a gerência automática da memória (GANGEMI, 2021). O garbage collector mantém uma contagem das referências para cada endereço de memória alocado pelo problema, de forma a liberar o espaço de memória quando a contagem de referências for nula. O garbage collector do Go é acionado automaticamente em alguns cenários, mas também é executado periodicamente pelo runtime do Go.

4.1.3 Interfaces

O Golang conta com uma definição chamada de "Interfaces". Interfaces são um conjunto de assinaturas de métodos. Assim, qualquer estrutura de dados que implemente todos os métodos da interface, automaticamente satisfaz a interface.

Dessa forma, é possível ter uma flexibilidade maior na hora de restringir os tipos dos campos de cada estrutura de dados. Por exemplo, no lugar de dizer que um

certo array é de um determinado tipo, é possível definir que o array é composto por elementos que implementam determinada interface.

Esse conceito se torna muito interessante para o uso de injeção de dependências no Golang.

4.1.4 Compilação

A linguagem Go visa diminuir ao máximo o tempo de compilação (reduzindo o número de otimizações no código, por consequência). O compilador do Go suporta diversas arquiteturas, como x86, x64, arm (v7, v8), MIPS etc. Assim, a cross-compilação de programas em Go tende a ser muito simples. Como todo executável é gerado linkando estaticamente todas as bibliotecas utilizadas, o executável resultante não tem nem ao menos dependências de sistema. Isso faz com que a linguagem seja muito atrativa para desenvolvedores de software embarcado.

Um exemplo de cross-compilação no Go, para a arquitetura armv7, é o seguinte:

```
GOOS=linux GOARCH=arm GOARM=7 go build .
```

Como o executável conta com diversas bibliotecas linkadas estaticamente, é possível que o tamanho do executável seja grande demais para algumas aplicações (como sistemas embarcados). Nesse caso, é possível utilizar ferramentas de compactação de executáveis como o UPX, que pode reduzir o tamanho de uma aplicação Go em até 70%, conforme métricas estimadas pelos próprios desenvolvedores do UPX.

4.2 COMUNICAÇÃO RPC

Segundo (GARCIA, 2021), o RPC (Remote Procedure Call) define um protocolo para execução remota de procedures em computadores ligados em rede. O RPC não define um protocolo de transporte específico. Assim, é possível que, numa arquitetura com múltiplos processos, esses mesmos processos possam se comunicar através TCP, UDP ou UDS.

No RPC, um elemento se comporta como cliente e o outro como um servidor. A comunicação entre esses elementos é sempre iniciada pelo cliente. O cliente faz então uma chamada de função remota e aguarda uma resposta, que pode ser uma informação ou um erro (o protocolo garante que, se houver falha na execução da rotina remota, é retornado um erro).

O fluxo RPC é detalhado na imagem a seguir:

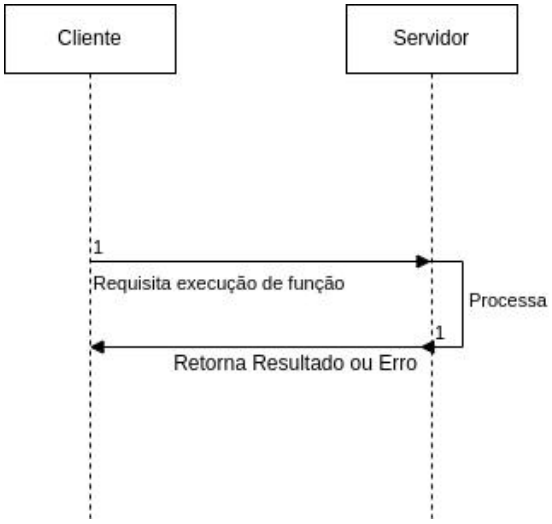


Figura 6 – Interação cliente-servidor RPC

4.2.1 JSON/RPC

O JSON/RPC é uma forma de comunicação RPC, onde o formato das mensagens trocadas entre o cliente e o servidor é o JSON. Assim, se estabelece uma convenção do formato da mensagem RPC, que não é especificado em clientes RPC normal. O fluxo de chamadas é o mesmo que o do RPC, a única diferença é que há uma padronização das mensagens utilizadas.

4.3 REDES MÓVEIS

Redes móveis são tecnologias de comunicação sem fio inicialmente desenvolvidas para transportarem aúdio de aparelhos celulares para operadoras. A partir da segunda geração dessa tecnologia, foi possível fazer também o envio de dados genéricos (como SMS) entre os aparelhos conectados. Em sua terceira geração, as taxas de transferência de dados chegaram ao ponto de permitir a naveção na internet pela rede móvel. Atualmente, no Brasil, a tecnologia já está na sua quarta geração, conhecida como 4G. Detalhes sobre as gerações de redes móveis são apresentadas a seguir. As informações nessa seção foram extraídas de (BHANDARIL; DEVRA; SINGH, 2017).

4.3.1 Evolução das Tecnologias Móveis

A primeira geração de redes móveis (1G) era analógica e contava com taxas de transmissão de até 2,4kbps, e foi projetada para transferir áudio apenas. A geração seguinte (2G) foi baseada numa tecnologia de transmissão digital, alcançando até 9,6kbps em sua primeira versão e capaz de transferir mensagens de texto. Ao longo do tempo o 2G evoluiu para taxas de transmissão de até 150kbps. A terceira geração (3G) se consolidou alcançando taxas de transmissão de até 2Mbps, o que fez com que aplicações dependentes da tecnologia móvel pudessem melhorar funcionalidades

que até então eram precárias, como a naveção na internet. A quarta geração (4G) compreende tecnologias como o Long Term Evolution (LTE), e o IEEE 802.16 (WiMAX), com taxas de transmissão entre 100Mbps e 1000Mbps.

4.4 LINUX EMBARCADO

Linux Embarcado consiste na utilização do Kernel Linux em sistemas embarcados, caracterizados por terem arquiteturas muito restritivas (em questão de capacidade de processamento, armazenamento e memória). De acordo com (TANENBAUM; AUSTIN, 2013), o Linux surgiu no início dos anos 1990, sendo desenvolvido por Linus Torvalds, com base no estudo de um sistema operacional chamado MINIX, disponibilizado por Andrew Stuart Tanenbaum para seus alunos da Universidade de Helsinki em 1987. O MINIX, por sua vez, foi uma implementação acadêmica de um sistema compatível com o UNIX, que por sua vez foi desenvolvido pela Bell Labs no início da década de 1970. Assim, há uma grande árvore de influências que culminaram no Linux. Todos esses sistemas operacionais mencionados implementam uma interface chamada Portable Operating System-IX (POSIX), padronizada pela Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE) na norma P1003.

Ao kernel do Linux foram acrescentados ferramentas da Free Software Foundation (FSF). O conjunto formado pelos utilitários e o kernel do Linux é chamado de GNU/Linux (ENCYCLOPAEDIA BRITANNICA, 2021).

4.4.1 Software Livre

Um dos grandes diferenciais do Linux, que faz com que ele seja atrativo para vários fabricantes, é sua licença de software permissiva, a GNU General Public License versão 2 (GPLv2). Essa licença, escrita por Richard Stallman para a FSF, faz com que o software protegido possa ser livremente distribuído, colocando como contrapartida que toda a modificação feita no código deve ser pública. Isso permite que diversos fabricantes tenham suas próprias distribuições Linux adaptadas ao próprio hardware, sem a necessidade de pagar uma licença pelo software.

4.4.2 Drivers

Drivers são utilizados para fazerem a comunicação entre o sistema operacional e equipamentos de hardware. No Linux, os equipamentos de hardware conectados são tratados como arquivos, de forma que a interação com o driver é simples. Em geral, para que um determinado equipamento possa operar corretamente com o sistema operacional, é necessário que o sistema tenha os drivers necessários para a comunicação com o equipamento. Muitas vezes esses drivers são providos pelo fabricante dos dis-

positivos, mas também existem drivers comunitários desenvolvidos pela comunidade de software open-source.

4.5 PPP

O Point to Point Protocol (PPP) é um mecanismo para criar e utilizar o Internet Protocol (IP) e outros protocolos de rede sobre um link serial, onde o link serial pode ser um modem conectado via USB, uma conexão telnet ou até mesmo uma linha telefônica.

O protocolo PPP é um protocolo peer to peer, de forma que não há nenhuma diferença entre a máquina que inicia a conexão e a máquina que aceita a conexão. No entanto, é comum se referir a essas máquinas como cliente e servidor (respectivamente), por motivos de clareza na explicação do protocolo.

Um computador com o sistema operacional Linux pode ser tanto um cliente quanto um servidor PPP, até mesmo simultâneamente. Não há limitação para a quantidade de conexões PPP ativas num mesmo computador.

Para utilizar o PPP no Linux, é necessário habilitar um módulo PPP no Kernel (de forma que o kernel seja compilado com suporte a tecnologia).

Ao discar e ter a conexão aceita, um cliente PPP no Linux gera uma interface de rede virtual no sistema – normalmente denominada "ppp0"(onde o número pode variar).

4.6 HTTP

O HyperText Transfer Protocol (HTTP) é o protocolo padrão para a troca de arquivos na internet. O HTTP foi construído sobre o protocolo TCP/IP. O HTTP, por padrão, utiliza uma arquitetura de cliente-servidor. A versão 1.0 do HTTP foi lançada em 1996, e tinha como principal característica o fato de que as conexões HTTP não mantinham estado, no sentido de que cada requisição de um cliente estabelecia uma nova conexão com o servidor. Em 1997 foi lançado o HTTP 1.1, que incluia as funcionalidades de conexões persistentes e múltiplos domínios com o mesmo endereço IP. Em 2015 foi lançado o HTTP 2, que utiliza um formato de serialização binário para as páginas web (no lugar do plaintext).

Nos anos 2010, muitos websites começaram a utilizar o HTTPS (HTTP seguro, com uma camada de criptografia SSL/TLS). A porta TCP padrão do HTTP é a 80, e a do HTTPS é a 443

O HTTP implementa diversos tipos de requisições, sendo as mais importantes o POST, GET, PATCH e DELETE, que implementam o básico de um Create, Read, Update, Delete (CRUD). Assim, para cada endereço, uma requisição pode ser de um tipo diferente e, para cada tipo, o tratamento da requisição pode ser diferente. Em geral,

o POST é responsável por incluir novas informações. O GET é responsável por retornar informações do servidor. O PATCH pode ser implementado para atualizar informações do servidor, enquanto o DELETE permite remover informações do servidor (Figura 7).

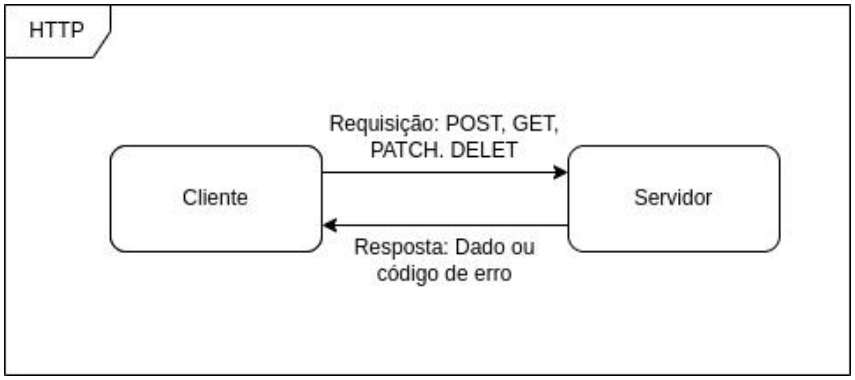


Figura 7 – Arquitetura cliente-servidor HTTP

Com base no protocolo HTTP/HTTPS, pode ser implementada uma Application Protocol Interface (API) para realizar as operações de CRUD no servidor. É com base nesse protocolo que a internet pôde crescer e se expandir, então seu valor é incomensurável.

4.7 MQTT

O MQTT é um protocolo baseado no TCP/IP, com suporte a criptografia utilizando SSL/TLS e autenticação via usuário e senha. A porta TCP padrão para esse protocolo é a 1883, enquanto a 8883 é a padrão quando se utilizada o SSL/TLS.

Nesse protocolo, existem dois elementos principais: os clientes e o broker. Os clientes podem assumir dois papéis diferentes, de publisher e subscriber, ou ainda os dois simultâneamente. O papel de publisher é o papel de gerar informações e postá-las no broker. São dois os dados que o publisher disponibiliza para o broker, a informação em si e um tópico associado a informação. O tópico age como um filtro de informações. Dessa forma, um cliente do tipo subscriber age consumindo informações, e para isso, ele se inscreve em determinado tópico (para demonstrar para o broker o interesse em certo tipo de informação). Assim, o papel do broker se resume a intermediar a troca de informações entre clientes do tipo publisher ou subscriber, roteando as informações de forma adequada. Um broker, em geral, é projetado para suportar milhares de clientes simultâneamente.

A Figura 8 demonstra uma arquitetura simples MQTT, com um broker e três clientes, sendo um deles um publisher, um deles um subscriber e outro deles um publisher/subscriber.

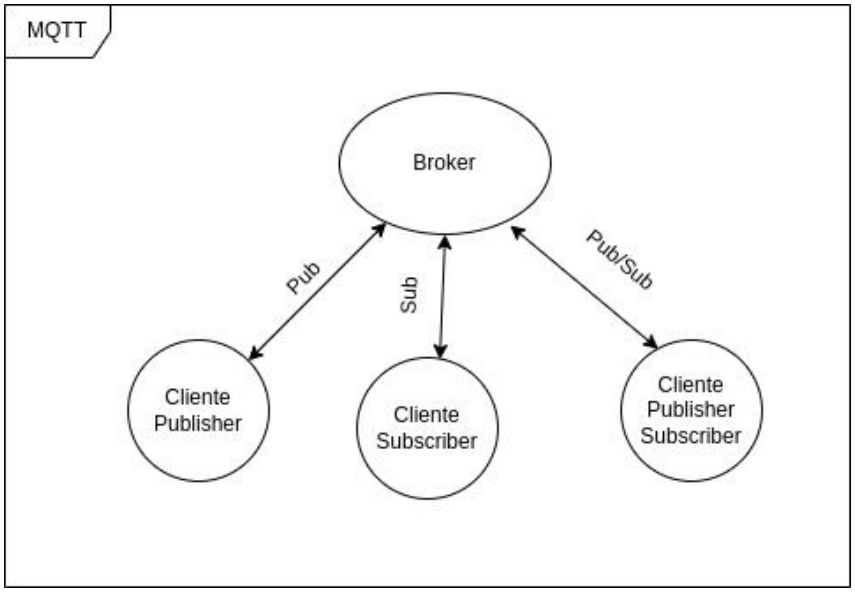


Figura 8 – Arquitetura cliente-broker MQTT

4.8 NTP

O Network Time Protocol (NTP), em tradução livre, Protocolo de Tempo para Redes, é um protocolo de rede que padroniza uma forma de sincronização entre relógios de dispositivos do tipo cliente com relógios de servidores (que agem como fontes para o horário real). Além do protocolo em si, o NTP estipula os algoritmos a serem utilizados na comunicação com os servidores, com a finalidade de calcular a diferença de tempo, estimar o erro, escolher as melhores referências e ajustar o relógio local.

Para realizar a sincronização, o NTP constrói uma rede lógica dinâmica, com características de uma rede peer-to-peer.

4.9 OPENVPN

Uma Virtual Private Network (VPN), rede virtual privada, consiste num mecanismo seguro e encriptado para conectar um dispositivo a uma rede privada através de uma rede pública, como é o caso da internet. Ao utilizar um serviço de VPN, um usuário ganha a possibilidade de acessar remotamente um recurso compartilhado, sem a necessidade de expor esse recurso a internet como um todo (evitando toda uma série de ataques possíveis).

O OpenVPN é uma implementação de um cliente e de um servidor para VPN. Esses softwares, utilizados em conjunto, permitem todas as funcionalidades citadas no parágrafo anterior. O OpenVPN mantém uma licença permissiva e é, por conta disso, uma excelente escolha de VPN para empresas que não desejam investir muitos recursos em infraestrutura de redes.

4.10 METOLOGIAS ÁGEIS

Metodologias ágeis são uma resposta natural aos desafios modernos de gerência de projetos de tecnologia. Num ambiente de incertezas extremas (como é o caso das Startups e das empresas de tecnologia de ponta), um planejamento cascata pode ser extremamente difícil de ser bem estimado e seguido. Nesse contexto, o desenvolvimento do produto deve ser voltado a remover o maior número de incertezas o mais rápido o possível. Assim, as metodologias ágeis têm como princípios entregas pequenas (poucas funcionalidades por versão de software), para que os clientes tenham contato o quanto antes com essas entregas, validando se as funcionalidades desenvolvidas são realmente úteis. No caso positivo, vale a pena desenvolver e aprimorar essas mesmas funcionalidades. Caso a recepção seja negativa, a funcionalidade poderia ser rejeitada e descontinuada. Dessa forma, as metodologias ágeis são aplicadas de forma a minimizar os custos de desenvolvimento, evitando trabalho desnecessário pela equipe de produto. Existem algumas metologias que são consideradas ágeis, mas nessa seção apresentaremos apenas uma: o Kanban, que é o fluxo de trabalho recomendado para nossa unidade de negócios da Khomp.

4.10.1 Kanban

O Kanban é uma metodologia ágil, projetada para atender cenários em que os requisitos de uma solução são modificados com uma frequência muito grande, até mesmo diariamente. Assim, a metologia do Kanban permite que equipes se adequem rapidamente a novos requisitos.

Para projetos de IoT, essa metologia conta com uma sinergia excelente, devido ao ambiente de extrema incerteza característico de um mercado que ainda está se formando.

4.10.1.1 Características do Kanban

Um Kanban consiste em três elementos: um quadro com todas as situações possíveis para as tarefas de um determinado projeto; uma ficha representando uma tarefa (e as informações associadas); uma política de movimentação de fichas, para definir como as fichas se movimentam pelo quadro do Kanban.

As fichas são revisitadas diariamente, de forma que quaisquer mudanças de prioridade possam ocorrer num ciclo bem curto. Se as fichas atenderem as políticas de movimentação definidas, é possível de que um usuário do quadro possa executar a tarefa que melhor se adeque ao momento atual do projeto em desenvolvimento.

No quadro Kanban retratado na Figura 9, podemos ver uma série de fichas de tarefas distribuídas em três colunas: ”To Do”, ”Doing” e ”Done”. Um exemplo de política de movimentação de fichas para esse cenário seria:

- Para inserir uma ficha na coluna ”To Do”, é necessário especificar na ficha uma tarefa, a prioridade da tarefa e o grau de complexidade de execução;
- Para mover uma ficha de ”To Do” para ”Doing”, é necessário que seja atribuído um responsável técnico à ficha;
- Para mover uma ficha de ”Doing” para ”Done”, é necessário que o código associado a tarefa esteja devidamente testado e documentado;
- Para a remoção de uma ficha de ”Done”, é necessário registrar no changelog da versão de software que a tarefa foi concluída.

Esse tipo de regra define de forma estrita quais movimentações são possíveis. Movimentações não contempladas pela política de movimentação de fichas não são possíveis de serem realizadas.

Figura 9 – Exemplo de quadro Kanban

Fonte: Site Digité.

5 REQUISITOS PARA A SOLUÇÃO

Nesse capítulo serão apresentados os requisitos funcionais e não-funcionais para a solução de solução de software que implementará a nova gerência de conectividade do gateway.

5.1 REQUISITOS FUNCIONAIS

Os requisitos funcionais estão listados a seguir. Logo após a lista estão as explicações de cada item.

- Suporte ao modem Quectel 4G
- Suporte ao modem Telit 3G
- Conexão via protocolos internet com serviços em cloud
- Seleção de cartão SIM
- Configuração de APN para cada cartão SIM
- Configurações de rede cabeada
- Configurações de comportamento de fallback e failover das interfaces
- Validadores de interfaces de redes
- Interface Web com informações sobre conectividade e modem

5.1.1 Suporte ao Modem Quectel 4G

É esperado que a aplicação detecte automaticamente qual o modelo do modem, e utilize os comandos AT apropriados para o modelo em questão.

5.1.2 Suporte ao modem Telit 3G

É esperado que a aplicação detecte automaticamente qual o modelo do modem, e utilize os comandos AT apropriados para o modelo em questão.

5.1.3 Conexão via protocolos de internet com serviços em cloud

Com um modem disponível, deve ser possível estabeler conexão com a internet através desse modem.

5.1.4 Seleção de cartão SIM

Cada modem suporta dois cartão SIM. Espera-sa que seja possível escolher o cartão SIM padrão.

5.1.5 Configuração de APN para cada cartão SIM

Cada cartão SIM tem uma APN própria para conexão com a internet (host, usuário e senha). É esperado que seja possível cadastrar qualquer APN para uso com o cartão SIM.

5.1.6 Configurações de rede cabeada

Configurações do equipamento relacionadas a interface Ethernet. Permitir o uso de DHCP para configuração da rede do equipamento. Permitir o uso de um IP fixo, máscara de rede e gateway padrão para a interface de rede. Permitir o cadastro de servidores de DNS.

5.1.7 Configurações de comportamento de fallback e failover das interfaces

Deve ser possível definir qual a política de envio de dados para servidores na internet: exclusivamente via conexão cabeada, exclusivamente via conexão da rede móvel, via conexão cabeada com fallback para redes móveis. Além do fallback, no caso do modem também há o failover entre os cartões SIM.

5.1.8 Validadores de interfaces de redes

Esses validadores são elementos que permitem testar se uma determinada interface de rede está operacional.

5.1.9 Interface Web com informações sobre conectividade e modem

As informações desejadas são as seguintes:

- Fluxo de dados pela Ethernet
- Fluxo de dados pelo modem
- Status dos validadores associados a cada interface de rede
- Configurações em uso
- Nível de sinal do modem
- Tecnologia de rede móvel em uso

- Operadora da conexão móvel
- Cartão SIM em uso
- APN em uso
- Endereço IP das interfaces de rede

5.2 REQUISITOS NÃO-FUNCIONAIS

Os requsitios não-funcionais são os seguintes:

- Compatibilidade com os modelos de hardware ITG200 e ITG300
- Trocas de interface padrão de rede em um tempo menor que quinze minutos
- Comunicação síncrona com qualquer outro serviço do Gateway por meio de chamadas RPC

6 PROJETO DE SOFTWARE

Nesse capítulo é detalhado todo o desenvolvimento do projeto de software. Primeiramente, será apresentado o ambiente de trabalho utilizado, e em seguida, o fluxo para desenvolvimento da aplicação de gerência de conectividade, do Linux Embarcado e da interface web.

6.1 AMBIENTE DE TRABALHO

Para todo o desenvolvimento do projeto, foi utilizado um computador com o sistema operacional Manjaro Linux. O Manjaro é um sistema baseado em Arch que conta com milhares de repositórios para os mais diversos tipos de ferramentas. Dentro do Manjaro, foi criado um ambiente integrado de desenvolvimento. Cada componente desse ambiente está listado nas subseções a seguir.

6.1.1 Editor de Texto - Emacs com o Spacemacs

Como editor de texto, foi utilizado o Emacs com o conjunto de configurações conhecido como Spacemacs. O editor conta com diversos modos de comandos e modos de edição, e tem um excelente suporte aos mais diferentes plugins, como checadores de sintaxe, ferramentas de auxílio no desenvolvimento com a linguagem Go, auto-completes, integração com gerenciadores de versão, entre outros.

6.1.2 Versionamento de código - Git

Para o versionamento de código foi utilizado o Git, de acordo com os procedimentos internos da empresa descritos nos capítulos anteriores.

6.1.3 Compilação - Go

Para a compilação do código para a arquitetura ARM v7 utilizada no gateway, foi utilizado o compilador Go padrão, desenvolvido pela Google, na versão que implementa a linguagem como descrito na especificação do Go 1.17.

6.1.4 Softwares auxiliares

Nessa categoria entram os softwares que auxiliaram no desenvolvimento, mas que não são essenciais. Entre eles, se destacam os plugins do ZShell, que permitem o auto-complete no terminal. Além disso, cabe citar também os containers docker, utilizados para execução de testes.

6.2 INÍCIO DO DESENVOLVIMENTO

Com todos os componentes necessários para o desenvolvimento devidamente instalados na estação de trabalho, e com acessos aos equipamentos de hardware necessários (gateway ITG200, modem 3G, modem 4G, conversor usb-serial) e alguns cartões SIM de diferentes operadoras, o desenvolvimento foi iniciado. O primeiro passo, foi a validação de algumas premissas:

- Os serviços internos do gateway se comportam como esperado ao se utilizarem múltiplas tabelas de roteamento IP
- O modem 4G poderá ser administrado com o uso do PPP (tecnologia apresentada em capítulos anteriores)
- A comunicação serial com os modems poderá ser realizada através de alguma biblioteca do Go.

6.2.1 Validação das premissas

O primeiro teste realizado, e também a mais desafiadora, foi a validação de que os serviços internos do gateway se comportariam bem num ambiente de múltiplas tabelas de roteamento. Para isso, foi instalado um Linux Embarcado com suporte a múltiplas tabelas de roteamento em um gateway ITG200. Sobre esse Linux embarcados, foram instalados todos os pacotes de software utilizados pelo gateway. Após a inicialização de todos os serviços e interfaces de rede (modem 3G e Ethernet), os gerenciadores de rede foram desativados. Nesse momento, por via de comandos no terminal Unix, foi feita uma configuração manual das múltiplas tabelas de roteamento. Ao final dessa configuração, foram executados scripts de teste escritos em Python, para verificar se os clientes HTTP e MQTT utilizados pelo gateway conseguiam fazer bind com uma interface de rede e reconhecerem as tabelas de roteamento associadas com cada interface.

Essa primeira validação deu certo, os clientes puderam fazer o bind pelo endereço IP de cada interface. Assim, ao mesmo tempo, era possível ter um cliente HTTP em cada interface de rede disponível no gateway (o que não era possível anteriormente, pois antes das múltiplas tabelas de roteamento só havia uma rota padrão).

Essa validação com protocolos baseados em TCP foi a base para que fosse assumido que os demais protocolos baseados em TCP também poderiam utilizar as múltiplas tabelas de roteamento.

O próximo ponto avaliado foi a utilização do PPP para a criação de uma interface de rede virtual para o modem 4G. Para isso, bastaria verificar se o PPP era capaz de reconhecer o modem e gerar, no Linux, a interface de rede ppp0 (nesse momento ainda não seria necessário pegar IP com operadora, ou lista de DNS). Para isso, a aplicação

PPP foi cross-compilada para a arquitetura do gateway, e instalada ao lado das demais aplicações no gateway. O ponto de maior dificuldade nesse momento foi encontrar exatamente qual combinação de configurações do PPP permitiria a criação da interface virtual. Após algumas tentativas frustradas, foi verificado que uma das dependências do PPP estava numa versão abaixo da versão recomendada. Ao atualizar a aplicação em questão (chamada 'chat') o processo de criação da interface virtual ocorreu como esperado.

Por fim, a última validação foi a de que teria uma biblioteca serial para Golang capaz de se comunicar com os dois modelos de modem. Foram testadas três opções de bibliotecas, e delas, duas funcionaram sem muitos problemas. A biblioteca escolhida foi a que tinha menos efeitos colaterais (uma das bibliotecas gerava uma linha de log toda vez que recebia uma mensagem, e como não foi simples identificar como desativar esse comportamento, a outra biblioteca foi escolhida).

Tendo validado todos esses elementos, agora há uma certa segurança de que é possível escrever uma aplicação Golang para implementar todos os requisitos da solução. O capítulo a seguir detalha a implementação dessa aplicação.

6.3 APLICAÇÃO DE GERÊNCIA DE CONECTIVIDADE

Para o desenvolvimento dessa aplicação, foram executados alguns trabalhos principais, listados a seguir sem uma ordem particular:

- Implementação de abstrações para os comandos AT direcionados aos modems
- Receitas iniciais do Linux embarcado para cross-compilar dependências
- Configurações da aplicação via comando RPC e arquivo de configuração
- Implementação das regras de negócio com base numa máquina de estados
- Integração com o backend Python existente
- Interface Web

Esses trabalhos foram, muitas vezes, desenvolvidos paralelamente (por exemplo, a Interface Web foi sendo construída conforme novas funcionalidades foram surgindo) mas para fins didáticos, vamos ordenar a explicação de cada elemento pela ordem em que esses mesmos elementos ficaram prontos.

6.3.1 Comandos AT

A base de comunicação entre a aplicação e os modems são os comandos AT, feitos através de portas seriais. Para cada modem, existem comandos AT específicos

para configurar a APN, ativar o uso de SSL, iniciar conexão e obter informações. O formato de resposta dos comandos também é específico para cada fabricante.

Assim, foi definida uma interface do Golang com todos os métodos que uma classe de modems deveria ser capaz de implementar. Em tempo de inicialização, se define que a variável que representa os comandos do modem satisfaz a interface de "comandos", e esse objeto é inicializado em tempo de execução, de acordo com o modem detectado pelo gerenciador de conectividade.

Assim, podemos invocar métodos do tipo ”ReadCSQ” para ler o nível de potência do sinal, e essa mesma chamada funcionará para cada tipo de modem suportado (já que existe no objeto a implementação adequada do ”ReadCSQ”, usando a assinatura de método definida na interface).

Em resumo, a aplicação detecta o tipo de modem e inicializa um objeto que implementa as chamadas AT de acordo o modelo do modem, em tempo de execução.

6.3.2 Receitas para o Linux Embarcado

Foram definidas receitas para cross-compilação de aplicações Golang. Felizmente, um único comando para o compilador do Go já gera um binário compatível com a arquitetura ARM v7 do gateway. Em seguida, outro comando do UPX é utilizado para comprimir o executável resultante. Por fim, a receita copia os arquivos de configuração e o executável gerado para dentro do Linux embarcado.

Em qualquer momento do processo, em caso de erro, a receita exibe a mensagem de erro e interrompe o processo de build do Linux Embarcado.

6.3.3 Configurações via arquivo e RPC

Os arquivos de configurações são do tipo Tom’s Obvious, Minimal Language (TOML). Esse tipo de arquivo é legível por humanos, e tem um mapeamento direto para estruturas de dados do tipo "hash table", assim como o formato JSON, no entanto, o TOML tem a vantagem de também comportar comentários no arquivo de configuração.

Para a leitura do arquivo de configuração TOML foi utilizada uma biblioteca open source do Golang chamada Viper. O Viper permite ler uma série de formatos de arquivo de configuração (TOML, JSON, YAML, Dotfiles, etc) e acessar os dados de configuração de formas simples, com chamadas do tipo:

```
port := viper.Get("port")
```

No exemplo acima, o Viper analisa a estrutura do arquivo de configuração e retorna o primeiro valor que tenha como nome "port".

Para permitir uma estruturação melhor do programa, foram definidas estruturas de dados para comportar todos os campos dos arquivos de configuração. Assim, pode-

se utilizar um objeto com todas as informações definidas. O Viper permite isso através da notação:

```
var config MyConfigType
viper.Unmarshal(&config)
```

Com o uso dessa biblioteca, a administração dos arquivos se torna simples, pois também existem métodos para carregar, salvar e modificar arquivos.

Nesse momento do projeto também foi iniciado o desenvolvimento do servidor RPC para comunicação com as demais aplicações. A forma de se fazer a aplicação recarregar os arquivos de configuração foi através da chamada RPC de "HotReload". Quando o servidor RPC recebe um comando desse tipo, simplesmente usa o Viper para recarregar os arquivos de configuração.

6.3.4 Implementação das regras de negócio

A partir da comunicação serial com os modems, da possibilidade de leitura de arquivos de configuração e da implementação das chamadas RPC, foi iniciada a implementação das regras de negócio. Basicamente o que se espera nesse passo é que toda a configuração de rede com múltiplas tabelas de roteamento IP, que foi feita manualmente na etapa de validação das premissas, agora seja feita automaticamente pela aplicação. Para isso, foi definido um fluxo de inicialização da aplicação, conforme mostrado na Figura 10.

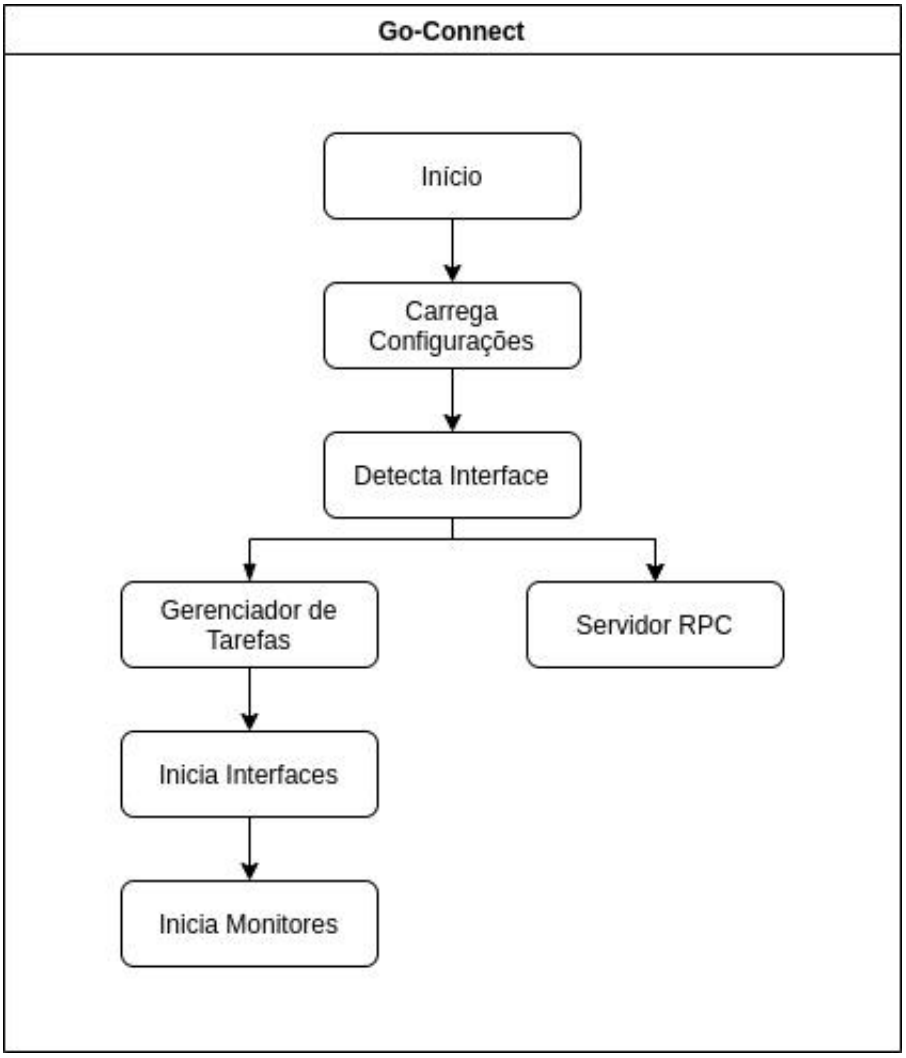


Figura 10 – Fluxo de inicialização da aplicação

A primeira tarefa executada pela aplicação é carregar os arquivos de configuração, que vão conter informações relevantes para o comportamento da aplicação em tempo de execução (por exemplo, se a política de fallback está ativa).

A partir do momento que essas informações estão disponíveis, a aplicação detecta automaticamente as interfaces de rede disponíveis (as cabeadas e as baseadas em redes móveis).

Em seguida, a aplicação disponibiliza o servidor RPC para receber comandos, e também inicia o gerenciamento de tarefas. O gerenciamento de tarefas inicializa as interfaces de rede (ou seja, as configura e as deixam prontas para uso). Por fim, é lançada a rotina que faz a verificação cíclica da conectividade em cada interface.

Com base nisso, detalharemos agora o fluxo específico de cada interface.

6.3.4.1 Ethernet

A configuração da Ethernet é simples. Se o DHCP estiver ativo, é lançado um daemon (aplicação de terceiros, que não é foco desse trabalho) para executar a configuração da interface. Em caso de configuração por IP fixo, são feitos comandos no Linux para configurar a interface de rede de acordo com as informações no arquivo de configuração.

O monitor da interface cabeada formato conforme representado na Figura 11:

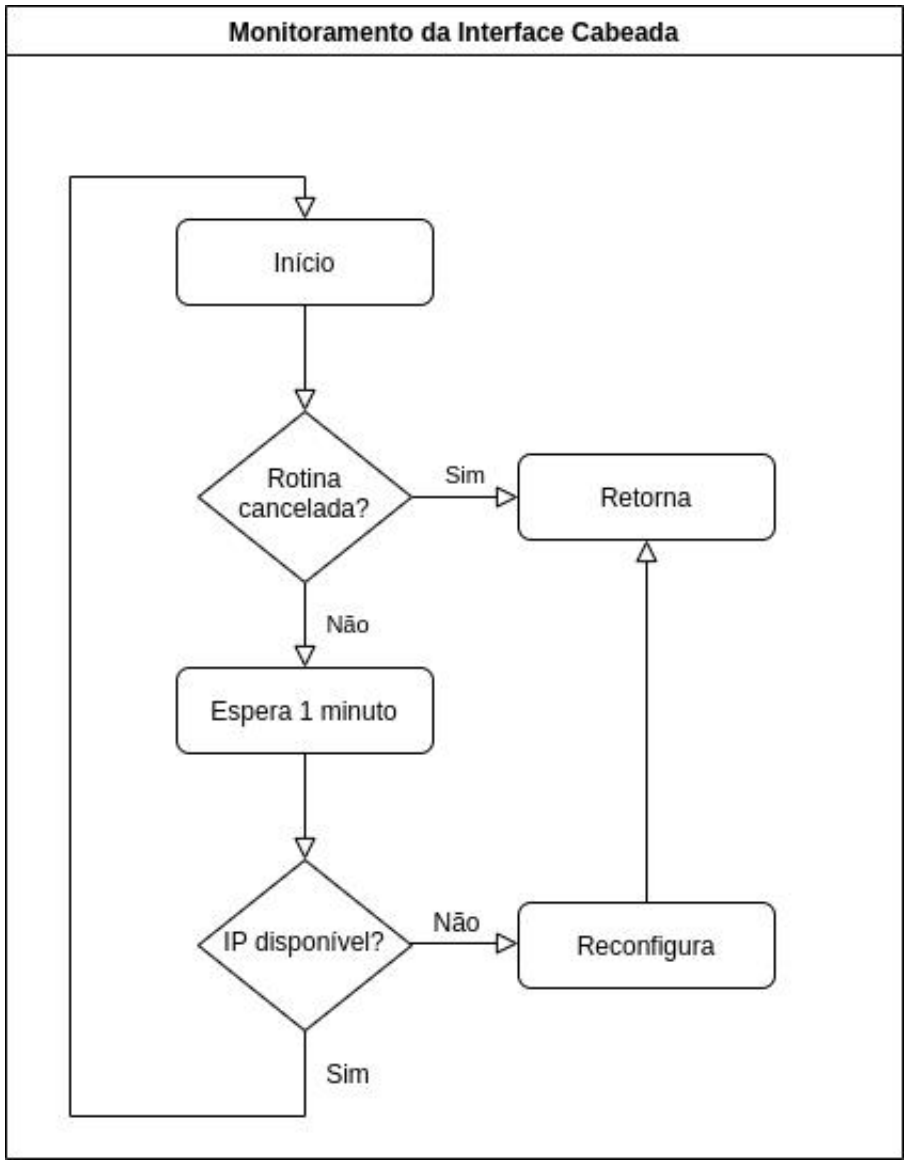


Figura 11 – Monitoramento da interface cabeada

Nessa caso, há um polling de um minuto para verificação da disponibilidade de um IP na interface. Caso não exista, a interface é reconfigurada. Caso a rotina de monitoramento seja cancelada, o monitor retorna.

6.3.4.2 Modem

A rotina de inicialização do modem é mais complexa do que a das interfaces cabeadas (pois vários aspectos da configuração dependem da comunicação com a operadora de dados associada ao cartão SIM em uso).

O modem primeiramente detecta os cartões SIM ativos, escolhe um deles para inicializar o procedimento de configuração e posteriormente realiza a configuração e conexão com a internet. Em caso de sucesso, é lançado o monitor do modem. Em caso de fracasso, é utilizada outra APN na configuração da APN. Esgotadas as APNs, em caso de failover ativo, a aplicação troca o cartão SIM em uso e inicia o procedimento novamente. Se após três tentativas o modem não realizar a conexão, a função de inicialização retorna um erro. Esse erro dispara uma nova tentativa de inicialização, após um certo tempo.

O diagrama da Figura 12 retrata o fluxo de inicialização do modem.

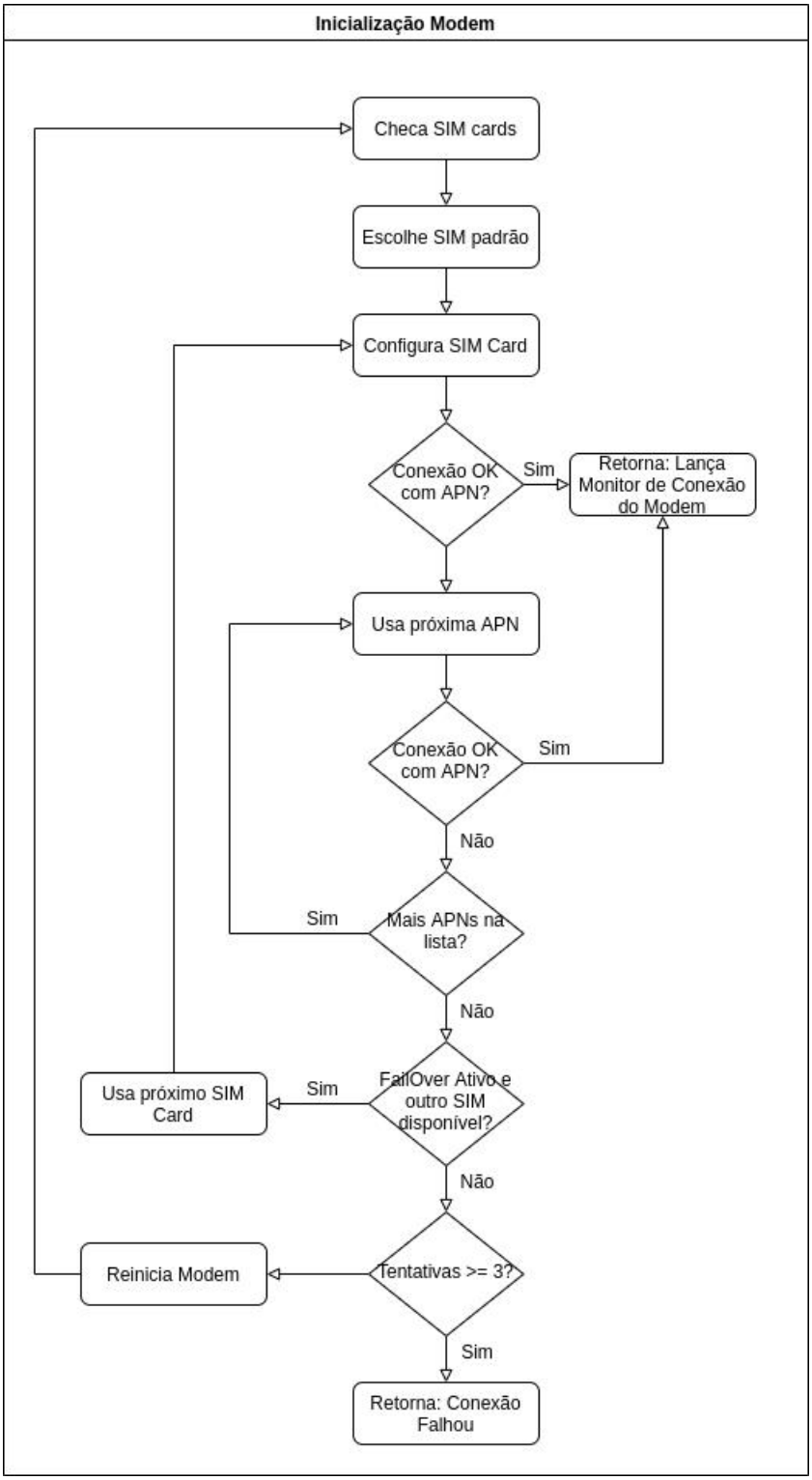


Figura 12 – Fluxo de inicialização do modem

A Figura 13 ilustra o fluxo da rotina de monitoramento da interface de rede associada ao modem.

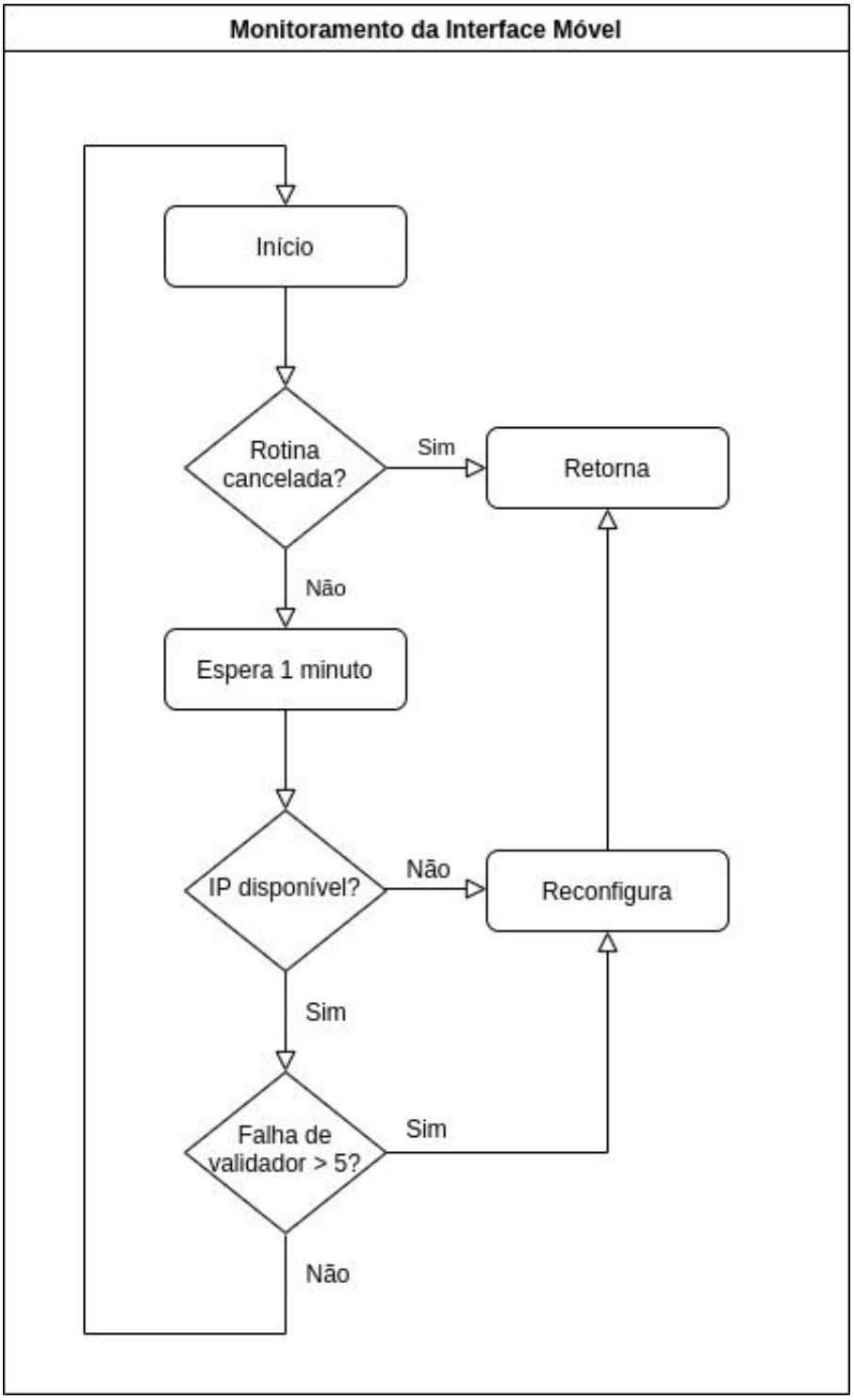


Figura 13 – Monitoramento da interface móvel

Seu funcionamento é análogo ao do monitor da interface cabeada, com o acréscimo de uma condição: no caso de 5 falhas sucessivas de conectividade, o modem é reconfigurado.

6.3.5 Integração com as demais aplicações do ITG200

Para realizar a integração da aplicação de gerência de conectividade com os demais serviços do gateway, o Go-Connect conta com um servidor RPC. A aplicação que anteriormente continha toda a lógica de negócios (ITG-Core) implementa um cliente RPC que solicita informações e envia comandos para o Go-Connect. Essa comunicação RPC entre as aplicações se dá sobre Unix Domain Sockets (UDS), que evita um pouco do overhead de comunicações TCP. O formato de descrição dos dados em cada mensagem é JSON.

O seguinte diagrama exemplifica o fluxo de chamados entre o cliente RPC do ITG-Core e o servidor RPC do Go-Connect (Figura 14).

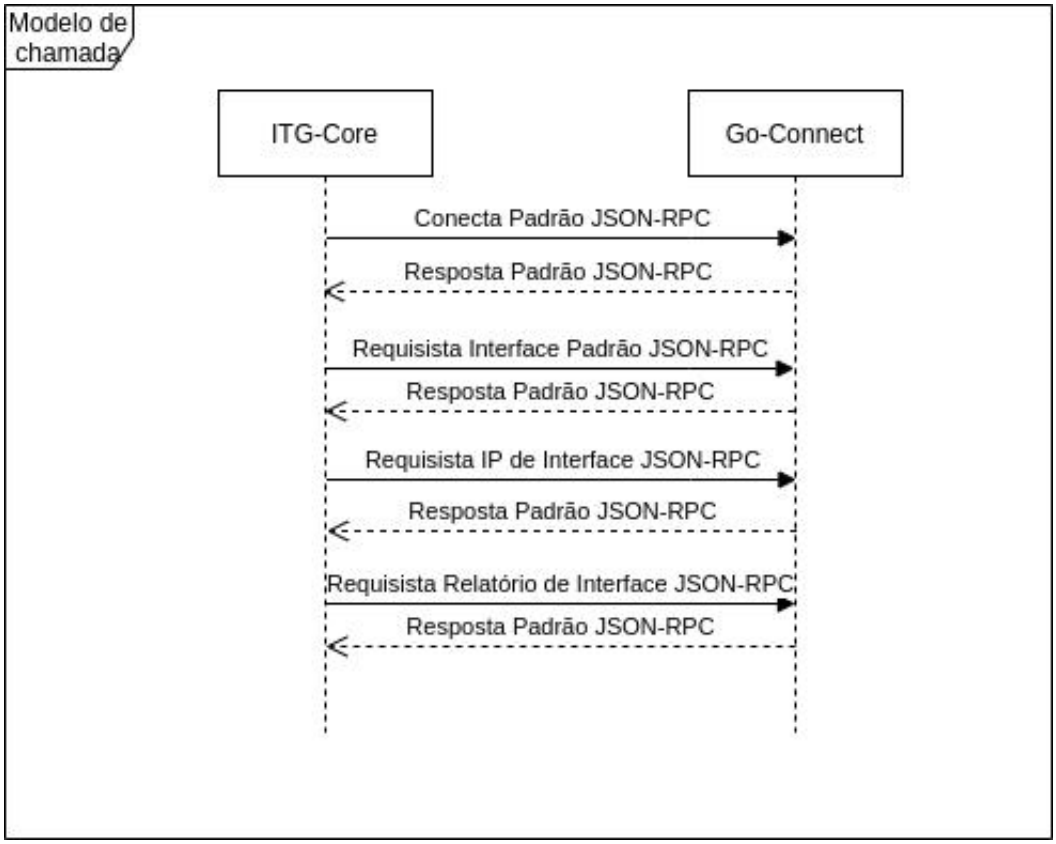


Figura 14 – Diagrama de chamadas - ITG-Core/Go-Connect

6.3.6 Interface Web

A interface web para a aplicação foi apenas uma expansão e adaptação da interface web já existente. Foram feitas novas telas para as configurações de conectividade, rede cabeada e rede móvel. Além disso, foram feitas telas de informações do sistema. Detalhes de cada tela gerada estão a seguir.

6.3.6.1 Configurações

Foram geradas três telas de configurações. A primeira delas (Figura 15) é para configurações gerais do comportamento da gerência de conectividade. Aqui estão as configurações de nível de log, política de conexão com a internet (apenas via interface cabeada, apenas via modem, ou interface cabeada com fallback pelo modem), e também os validadores de ping e de conexões TCP utilizados para que a aplicação possa determinar se está ou não com conectividade nas interfaces de rede.

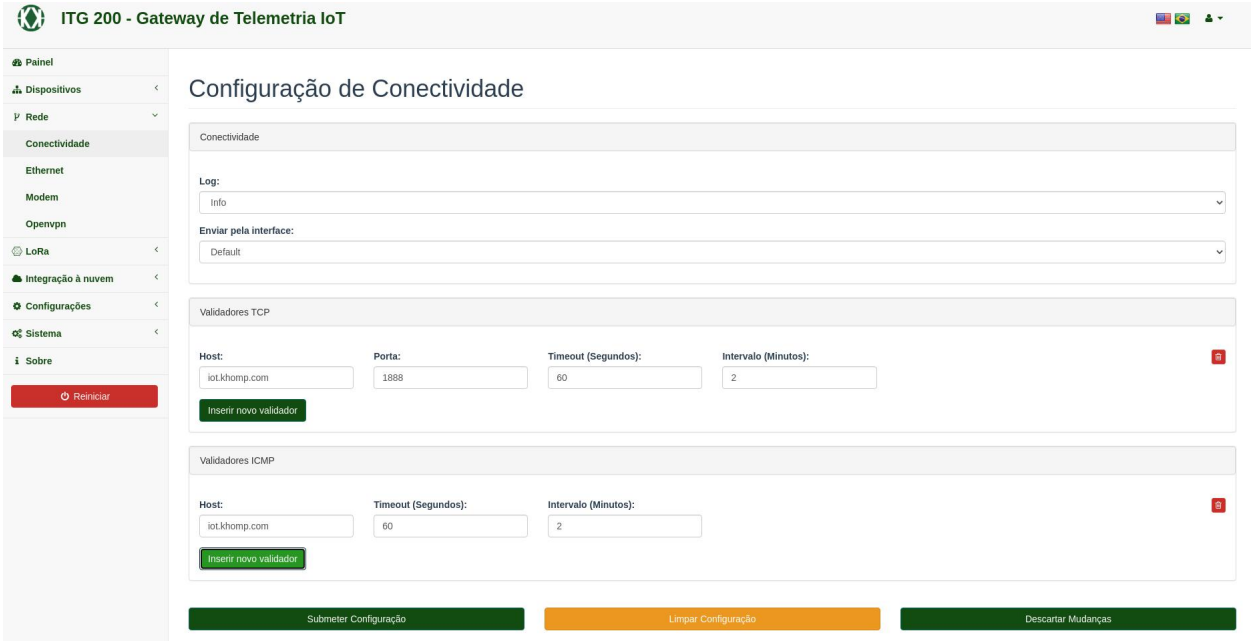


Figura 15 – Tela de configuração de conectividade

A segunda tela desenvolvida é a de configurações do modem. Nessa tela, é possível escolher o cartão SIM padrão, habilitar e desabilitar a política de failover entre os cartões SIM e configurar as opções de APNs para a conexão (Figura 16).

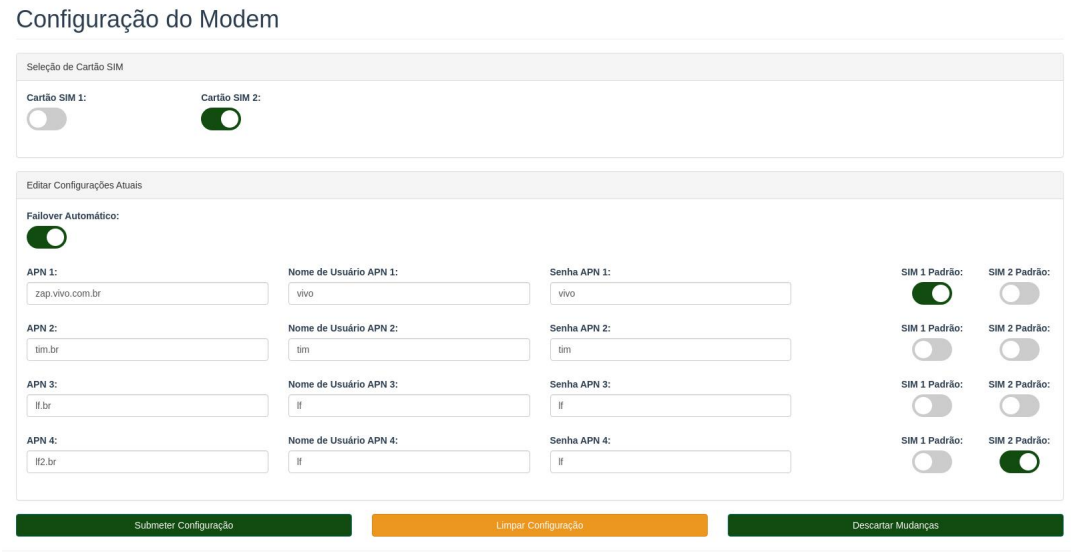


Figura 16 – Tela de configuração do modem

A terceira tela desenvolvida é de configurações da Ethernet. Nessa tela, é possível habilitar ou desabilitar o uso de DHCP, configurar um IP fixo, uma máscara de rede e um gateway padrão, além de ser possível de se registrarem servidores de DNS (Figura 17).

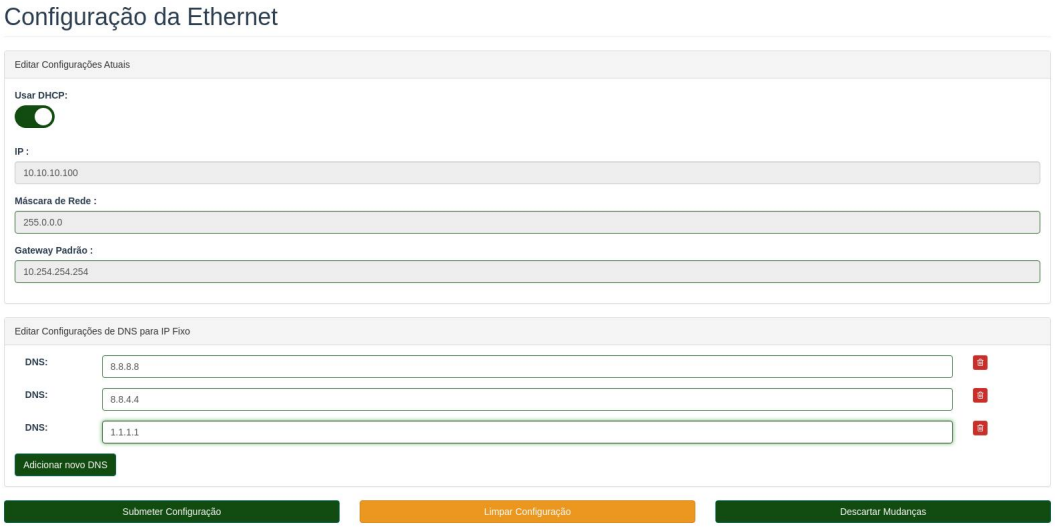


Figura 17 – Tela de configuração da Ethernet

6.3.6.2 Informações do Sistema

As telas de informações do sistema são as responsáveis por exibirem de forma simples as informações atuais de cada interface de rede e dos validadores de conectividade. A primeira parte dessa tela mostra os resultados dos validadores cadastrados, para cada interface de rede disponível no gateway (Figura 18).

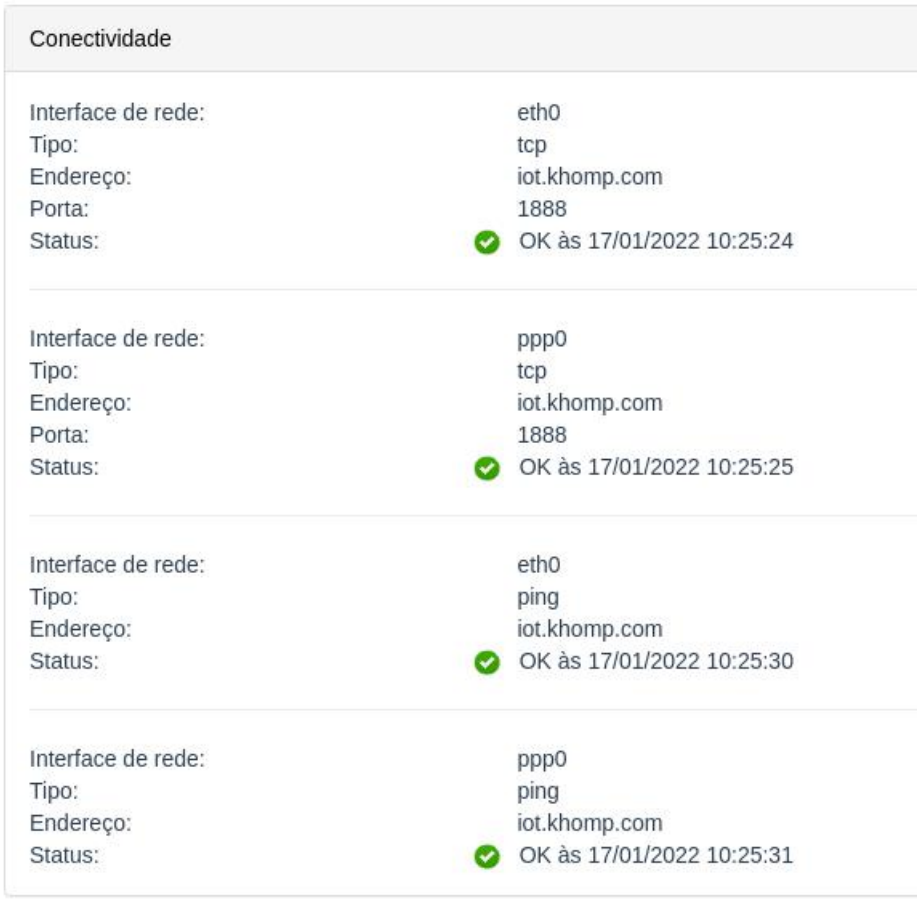


Figura 18 – Informações sobre validadores

A segunda parte dessa tela mostra as métricas de cada interface de rede, com a quantidade de bytes recebidos e enviados (Figura 19).

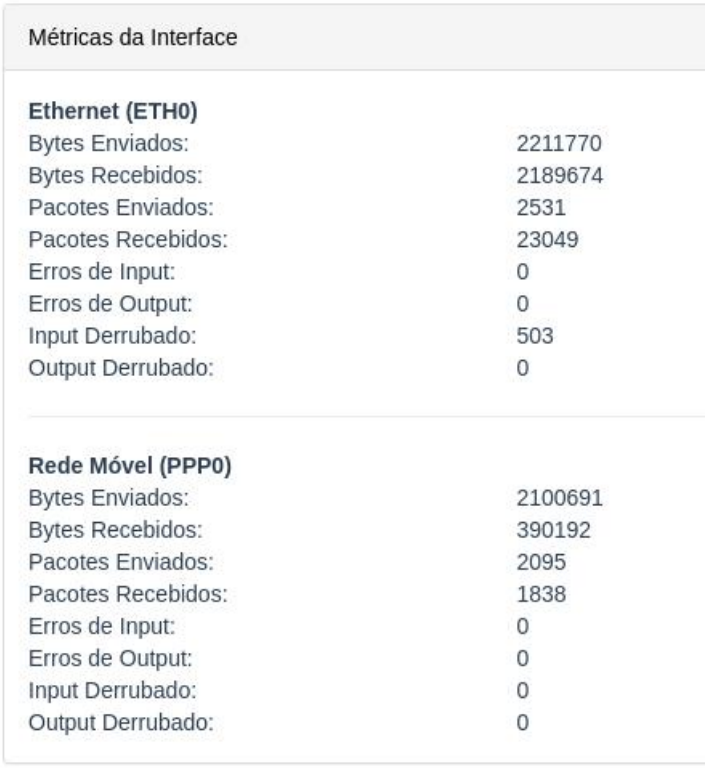


Figura 19 – Métricas das interfaces

A terceira parte da tela mostra todas as informações coletadas do modem (Figura 20).

Figura 20 – Informações do modem

Por fim, a última tela disponível exibe informações gerais do gateway (Figura 21).

Figura 21 – Informações gerais

6.3.6.3 Visualização de Logs

A aplicação Go-Connect gera informações disponíveis para o usuário por meio de arquivos de logs. Eles podem ser acessados via interface web, e inspecionados em casos em que uma análise do comportamento seja necessária.

6.4 ARQUITETURA DESENVOLVIDA

Com todos esses novos elementos de software implementados, temos uma nova imagem representando a arquitetura de software do ITG200. Essa nova arquitetura suporta modems 3G e 4G, conta com a aplicação Go-Connect para gerência de interfaces de rede e desacopla do ITG-Core todas as decisões relativas a conectividade. A Figura 22 ilustra a nova arquitetura do software do ITG200.

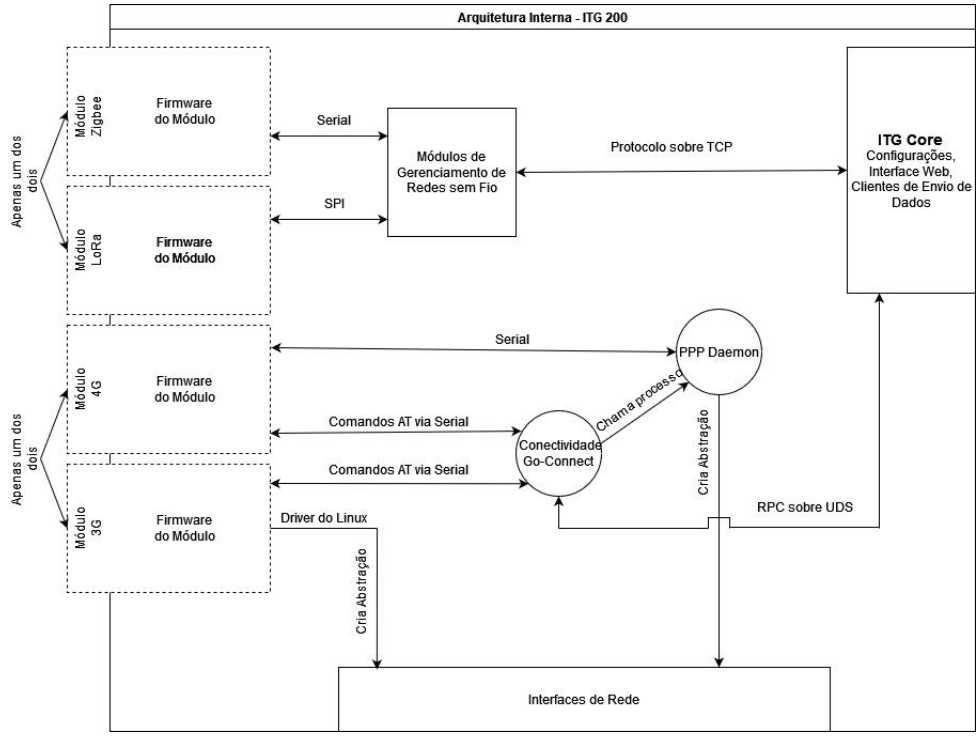


Figura 22 – Arquitetura do gateway com os novos elementos de software

7 ANÁLISE DOS RESULTADOS

Nesse capítulo serão discutidos os resultados obtidos pela nova gerência de memória, ao final do processo de desenvolvimento e validação por parte do QA. As métricas a seguir são derivadas das validações internas da Khomp, então se espera que, quando essa solução for implantada em campo, alguns bugs sejam descobertos. Nesse caso, caso sejam encontrados problemas em campo, patches corretivos deverão ser lançados. No que tange o presente trabalho, o desenvolvimento foi dado como concluído.

7.1 RESULTADOS ESPERADOS

Com base nos objetivos e requisitos do projeto, podemos definir os seguintes resultados como os esperados:

- Suporte ao modem 3G Telit
- Suporte ao modem 4G Quectel
- Uso simultâneo de interfaces de rede
- Reconexões e trocas de interface de rede padrão no menor tempo possível (preferencialmente menor que os quinze minutos de versões de software anteriores)

As três primeiras métricas são facilmente aferíveis, pois são métricas binárias, de sim ou não.

A métrica que tem a maior margem de interpretação é a última. Para esse trabalho, classificaremos os tempos de transição de interfaces da seguinte forma:

- Transição mais lenta que quinze minutos: ruim
- Transição entre oito e quinze minutos: regular
- Transição entre um e oito minutos: bom
- Transição na ordem de segundos: excelente

7.2 TESTES DE QA

Para aferir o funcionamento das funcionalidades desenvolvidas, a equipe de QA IoT da Khomp realizou testes extensivos sobre o equipamento utilizando a nova versão de software. Para isso, a equipe de QA tinha a disposição quatro gateways: dois do modelo IEEE 802.15.4, dois do modelo LoRaWAN. Para cada dupla de cada modelo,

um dos gateways manteve o modem 3G, e o outro recebeu um upgrade de hardware para utilizar o modem 4G.

Posteriormente, foram feitos testes dos gateways sem o uso de modems também. Detalhes sobre os testes realizados estão a seguir.

7.2.1 Roteiro de Testes

Para cada gateway com cada modelo de modem, foram feitos os seguintes testes, primeiro forçando o uso da interface Ethernet, posteriormente forçando o uso da interface móvel:

- Envio de dados via HTTP
- Envio de dados via MQTT
- Sincronização de data e hora via NTP
- Conexão OpenVPN
- Variações nos arquivos de configurações

Para o caso dos gateways LoRa, também ocorreram os seguintes testes:

- Envio de dados para servidor de rede LoRa via UDP

Em cada um desses testes, foram verificados os arquivos de configurações, o comportamento real do gateway, os logs gerados e as informações exibidas na interface web. Também foram extraídas métricas sobre transição de interfaces de rede.

7.2.2 Resultados Obtidos

Foram avaliados os envios de dados HTTP, MQTT, NTP, OpenVPN e UDP por todos os serviços descritos acima, tanto via interface cabeada, quanto por modem 3G e por modem 4G.

Ao final da homologação, todos os serviços estavam funcionando de forma campatível com os dois modems e com a interface cabeada. Isso valida a compatibilidade dos modems e das configurações de rede cabeada.

Os items podem ser avaliados da seguinte forma:

- Suporte ao modem 3G Telit: sim
- Suporte ao modem 4G Quectel: sim
- Uso simultâneo de interfaces de rede: sim
- Reconexões e trocas de interface de rede: menos que três minutos

Com base nisso, nas configurações de fábrica, o gateway agora realiza transições de rede de menos de três minutos (no pior dos casos). Esse resultado é considerado bom pela equipe de QA, pois é mais veloz que as transições de rede em versões anteriores de software.

7.3 TESTES DE CAMPO

Ao final da validação pelo QA IoT da Khomp, o produto será testado por parceiros integradores, para que eles possam ajudar a validar o produto em cenários reais. Essa nova camada de testes, medições e correções foge do escopo do presente trabalho, que já foi considerado concluído quando foi dado o aval do produto por parte da equipe de QA.

8 PROPOSTAS DE MELHORIAS

Neste capítulo serão apresentadas propostas de melhorias para a solução desenvolvida no ITG200. As propostas são divididas em duas seções: "otimizações internas", onde são propostas melhorias que não são imediatamente aparentes para os usuários do produto, e "novas funcionalidades", onde são apresentadas funcionalidades que não poderiam ter sido implementadas em versões anteriores do ITG200.

8.1 OTIMIZAÇÕES INTERNAS

Um dos aspectos mais importantes para uma boa gerência de conectividade são transições de interfaces de rede rápidas, ao ponto de que o ideal é que seja imperceptível para o usuário caso a transição ocorra, já que não haveria impacto em seu fluxo de trabalho. As primeiras propostas de otimizações são voltadas para isso. Além disso, como melhora de processo interno, é possível aumentar a cobertura de testes automatizados.

8.1.1 Comunicação Assíncrona

Atualmente, o motivo que faz com que o pior tempo de troca de interfaces seja de três minutos (e não instantâneo) é o fato de que a comunicação entre a aplicação Python, com a lógica de negócios, e a aplicação Golang, com a gerência de conectividade, se dá por meio de polling. Isso faz com que mudanças no status de conectividade sejam percebidos apenas no próximo polling. Isso ocorre porque a estrutura de comunicação RPC contém elementos com papéis bem definidos, de cliente e servidor. Assim, como a gerência de conectividade age como servidor, é necessária uma requisição por parte do cliente. A estratégia de polling não é a ideal, pois sempre vai apresentar um atraso. Para o propósito da primeira versão do produto, o fato do tempo de transição já ter sido reduzido basta, mas conhecendo as limitações da tecnologia podemos propor outros métodos para que a aplicação Python consiga saber que houve uma mudança no status de rede. É nesse ponto que entra a comunicação assíncrona, numa arquitetura de cliente-broker.

Como o gateway já conta com um broker MQTT interno, esse protocolo se transforma numa ótima opção para troca de mensagens assíncronas entre os processos (mesmo com o overhead associado ao protocolo) pois o tempo de desenvolvimento é significativamente menor do que outras técnicas possíveis (como a utilização do ZeroMQ).

Com base nisso, a proposta é integrar um cliente MQTT na aplicação Golang, de forma que a aplicação possa ativamente sinalizar suas mudanças de status para todo o sistema, através do broker MQTT interno. Assim, todos os processos interessados nas

mudanças da gerência de conectividade podem se inscrever em um tópico específico para essa finalidade, de reportar mudanças de interface de rede padrão.

Aplicando essa técnica é esperado que a transição de interfaces de rede seja quase instantânea (num tempo praticamente insignificante).

8.1.2 Serialização de Mensagens

O formato das mensagens que o Go-Connect troca com a aplicação Python é um JSON. O JSON é um formato legível para humanos, e portanto, contém cada caracter das mensagens codificado em UTF-8. Uma forma de diminuir o tamanho das mensagens que trafegam pelo gateway é trocar o formato legível por um formato binário, que compacte ao máximo a informação.

Existem algumas alternativas de métodos para serialização de mensagens, como os Protocol Buffers, o CBOR e o MessagePack. Destes, as alternativas mais simples de serem integradas ao gateway são as duas últimas, pois tem implementações com a mesma assinatura de métodos que a implementação do JSON no Python. Assim, o trabalho de trocar a formatação de JSON para CBOR consiste em instalar a biblioteca CBOR e trocar as chamadas do tipo "json.dumps()"por "cbor.dumps()"(nesse cenário específico).

O ganho aqui se deve ao fato de que mensagens menores tem um custo menor para serem transmitidas, e o custo de serialização é praticamente o mesmo.

8.1.3 Cobertura de Testes

Outra oportunidade de melhoria interna diz respeito ao processo de desenvolvimentos: é possível aumentar a cobertura de testes automatizados no produto. Existem vários tipos de testes que podem ser incluídos, como testes unitários, de integração e de aceitação. Cada um deles toca numa camada diferente da solução. O primeiro garante que a implementação de cada método está de acordo com os objetivos do negócio, o segundo garante que os módulos diferentes do sistema estão em harmonia, e o terceiro atesta que as características desejadas pelos usuários estão sendo atendidas.

8.2 NOVAS FUNCIONALIDADES

Além das melhorias internas, seja de software ou de processo, novas funcionalidades são possíveis agora que o gateway implementa as múltiplas tabelas de roteamento. Todas as possíveis funcionalidades aqui descritas não eram possíveis em versões anteriores do produto, pela forma como a gerência de conectividade original foi concebida.

8.2.1 Troca de Interface Instantânea

Com a nova gerência de conectividade, é possível implementar um cliente MQTT or interface web. Utilizando um mecanismo de sincronização, é possível alternar de um cliente para outro de forma instantânea. Essa funcionalidade poderia ser uma funcionalidade opcional (devido ao uso de dados móveis, manter uma conexão TCP aberta gera um custo).

8.2.2 Envio de Dados por Todas as Interfaces

Uma ferramenta de redundância possível com a nova arquitetura é a de estabelecer um cliente MQTT/HTTP por interface e mantê-los ativos simultâneamente, de forma que numa troca de interface de rede padrão, não haja chance de perda de informações.

8.2.3 Configurar clientes mobile-only

Outra possibilidade com a nova arquitetura é manter o envio de dados para um servidor em LAN através de uma interface de rede, e as interações com servidores na internet por meio do modem. Dessa forma, em cenários com a estrutura de TI restritiva, a quantidade de liberações que o gateway demanda na rede interna é mínima.

9 CONCLUSÃO

O projeto foi encerrado após uma série de testes bem sucedidos no QA IoT. Os resultados foram satisfatórios: todos os objetivos propostos no início do projeto foram alcançados. Além das vantagens de desempenho e flexibilidade, a nova implementação da gerência de conectividade se provou mais moderna e mais simples de ser compreendida e expandida. Sem dúvidas, a nova aplicação de gerência de conectividade se torna uma base sólida para novas funcionalidades no gateway.

Pelo fato de ser uma aplicação completamente isolada do restante da lógica de negócios do gateway, há uma flexibilidade maior para a equipe de desenvolvimento, que está livre para modificar as características internas das aplicações do gateway, tendo como única responsabilidade a manutenção da assinatura dos métodos de RPC, para garantir a compatibilidade do software. Isso contrasta com problemas anteriores de ”efeitos colaterais” quando haviam mudanças no processo de estabelecimento de conexões com a internet via rede móvel. A arquitetura final do gateway se mostra mais limpa, com os componentes melhor isolados.

Com a implementação do suporte a tecnologias de 4G, a Khomp expande os casos de uso de seu produto ITG 200. Há também uma sinergia com tecnologias de LTE que a Khomp comercializa. Isso permite propostas comerciais em que os elementos a venda são mais integrados, já que os gateways ITG 200 4G foram validados com tecnologias de LTE distribuídas pela Khomp.

REFERÊNCIAS

BHANDARIL, Nikhil; DEVRA, Shivinder; SINGH, Karamdepp. Evolution of Cellular Network: From 1G to 5G. **International Journal of Engineering and Techniques**, v. 3, p. 98–105, 5 2017.

BORGIA, Eleonora. The Internet of Things vision: Key features, applications and open issues. **Computer Communications**, v. 54, p. 1–31, 2014.

DRAKE, Joshua. **Linux PPP HOWTO**. 2000. Disponível em: https://tldp.org/HOWTO/PPP-HOWTO/. Acesso em: 11 jan. 2022.

ENCYCLOPAEDIA BRITANNICA, The Editors of. **HTTP**. [*S.l.*]: Encyclopædia Britannica, inc., 2022. Disponível em: https://www.britannica.com/technology/HTTP.

ENCYCLOPAEDIA BRITANNICA, The Editors of. **Linux**. [*S.l.*]: Encyclopædia Britannica, inc., 2021. Disponível em: https://www.britannica.com/technology/Linux.

GANGEMI, Scott. **An overview of memory management in Go**. 2021. Disponível em: https://medium.com/safetycultureengineering/an-overview-of-memory-management-in-go-9a72ec7c76a8. Acesso em: 15 jan. 2022.

GARCIA, Luís Fernando Fortes. **RPC - Remote Procedure Call**. 2021. Disponível em: http://penta.ufrgs.br/rc952/trab1/rpc.html. Acesso em: 16 jan. 2022.

GOOGLE. **The Go Programming Language Specification**. 2021. Disponível em: https://go.dev/ref/spec. Acesso em: 15 jan. 2022.

GUPTA, Prerna. **Device Drivers in Linux**. 2021. Disponível em: https://www.geeksforgeeks.org/device-drivers-in-linux/. Acesso em: 10 jan. 2022.

JEFFEY, Toby. **MQTT and CoAP, IoT Protocols**. 2014. Disponível em: https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php. Acesso em: 23 jan. 2022.

KHAN, R.; KHAN, S. U.; ZAHEER, R.; KHAN, S. Future Internet: The Internet of Things Architecture, Possible Applications and Key Challenges. **2012 10th International Conference on Frontiers of Information Technology**, p. 257–260, 2012.

OBERHUMER, Markus F.X.J.; MOLNÁR, László; REISER, John F. **UPX, the ultimate packager for eXecutables**. 2020. Disponível em: `https://upx.github.io`. Acesso em: 16 jan. 2022.

STALLMAN, Richard. **Licença Pública Geral GNU, versão 2**. 1991. Disponível em: `https://www.gnu.org/licenses/old-licenses/gpl-2.0.html`. Acesso em: 13 jan. 2022.

SUEIRO, Diego. **Yocto Project: Definições e Conceitos**. 2016. Disponível em: `https://www.embarcados.com.br/yocto-project-definicoes-e-conceitos/`. Acesso em: 17 jan. 2022.

TANENBAUM, Andrew S. **Computer networks**. 4. ed. [*S.l.*]: Prentice Hall, 2002. ISBN 0130661023,9780130661029.

TANENBAUM, Andrew S.; AUSTIN, Todd. Organização Estruturada de Computadores. *In*: 6. ed. [*S.l.*]: Pearson, 2013. P. 380–386.