Desenvolvimento de uma Plataforma VoIP de alta capacidade de chamadas

Caio Eduardo Borges Assis

Universidade Federal de Uberlândia
Faculdade de Computação
Bacharelado em Sistemas de Informação

Uberlândia
2019

Caio Eduardo Borges Assis

Desenvolvimento de uma Plataforma VoIP de alta capacidade de chamadas

Trabalho de Conclusão de Curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia como parte dos requisitos para a obtenção do título de Bacharel em Sistemas de Informação.

Área de concentração: Sistemas de Informação

Orientador: William Chaves de Souza Carvalho

Uberlândia
2019

Resumo

O documento descreve o desenvolvimento de uma Plataforma VoIP de alta capacidade de chamadas, cujo objetivo é fornecer um sistema capaz de centralizar a telefonia de uma empresa e fornecer usabilidade através de APIs disponilizadas pela arquitetura RESTful implementada pela aplicação. Desenvolvido com *softwares* de grande uso comercial, como Django, MySQL e FreeSwitch, o sistema permite adições e customizações, podendo ser instalado para gerenciar grandes fluxos de ligações.

Palavras-chave: VoIP. Internet. Comunicação.

Lista de ilustrações

Lista de tabelas

Lista de siglas

API Application Programming Interface

ARPANET Advanced Research Projects Agency Network

ATA Adaptador para Telefone Analógico

HTTP Hypertext Transfer Protocol

IAX Inter Asterisk Exchange

IP Internet Protocol

PABX Private Automatic Branch Exchange

REST Representational State Transfer

SIP Session Initiation Protocol

URL Universal Resource Locator

VoIP Voice over IP

XML Extensible Markup Language

Sumário

Capítulo 1

Introdução

Diante da crescente demanda por tecnologias de comunicação, o mercado possui uma diversidade de soluções the suprem, quase sempre parcialmente, esta demanda. A comunicação de voz pela Internet surgiu na década de 1990, mas somente no início dos anos 2000 que o VoIP (*Voice over Internet Protocol*) aumentou significativamente sua presença nos meios residenciais e comerciais. A adesão a este tipo de comunicação representa uma economia considerável em relação à telefonia convencional. Logo as empresas de Call-Center perceberam a viabilidade e adequação da tecnologia para seus propósitos. Por ser um meio de comunicação que compartilha de recursos com as aplicações WEB, logo apareceram *softwares* decidados a gerenciamento de chamadas.

1.1 A História do VoIP

A telefonia VoIP é uma tecnologia considerada recente. Conhecida por nomes como *Voice Over Broadband* (VoBB), *Internet Telephony*, *IP Telephony*, e *broadband Phone*, foi somente com os avanços da Internet que o VoIP pôde se consolidar e inovar o setor de comunicações (VOIP-INFO, 2019).

O e-mail, meio de comunicação consolidado na época, não satisfazia a vontade das pessoas de ter uma maneira de se comunicar de forma pessoal, em tempo real, e estas vontades foram algumas das motivações para a criação do VoIP.

Desenvolvido por volta de 1995, o VoIP tinha se tornado a segunda opção para realizar chamadas. Em ambientes comerciais, a utilização do VoIP acarretou em consideráveis economias nos gastos com telefonia. Para usuários comuns, surgiram aplicações capazes de utilizar a tecnologia para conectar um usuário ao outro, como, por exemplo, o Skype e Viber. O uso da tecnologia para realizar ligações para telefones convencionais se limitou ao ambiente comercial, devido ao aumento de pessoas que possuem *smartphones* e quedas nos valores de ligações nas operadoras de telefonia convencional, e os usuários comuns continuavam utilizando o Skype e outros aplicativos de comunicação.

1.2 Motivação

Devido à crescente demanda por soluções de telefonia robustas no ambiente comercial, surgiu a necessidade por *softwares* capazes de processar e gerenciar altos volumes de chamadas simultâneas.

1.3 Objetivos e Desafios da Pesquisa

Desenvolver o *backend* de uma plataforma VoIP que implementa a arquitetura RESTful (MOZILLA, 2019a) e fornece uma coleção de APIs a fim de gerenciar configurações de usuários, regras de roteamento, registro de ligações e com capacidade de lidar com grandes volumes de chamadas, utilizando *softwares* de código aberto.

1.4 Hipótese

A contínua e crescente demanda por capacidade de operacionalização de chamadas por grandes empresas e, notadamente, por empresas de Call-Center, esta pesquisa propõe que a criação do *backend* de uma plataforma VoIP de alta capacidade de chamadas, que fornece recursos pelo menos compatíveis com aqueles verificáveis atualmente no mercado de servidores VoIP e que atende, sobretudo, demandas de serviços de chamada para Call-Centers e grandes empresas.

1.5 Contribuições

Esta pesquisa contribuirá tanto com a área de pesquisa sobre VoIP, quanto com a área de desenvolvimento de soluções, ao fornecer uma arquitetura RESTful para criação de aplicações de gerenciamento de grandes volumes de chamadas VoIP capazes de atender a usuários exigentes.

1.6 Organização do Trabalho

Este trabalho será organizado da seguinte maneira: O Capítulo 1 trata da Introdução ao tema, o capítulo 2 apresenta os fundamentos, conceitos e trabalhos correlatos ao tema, o Capítulo 3 apresenta o desenvolvimento da plataforma e, por fim, o Capítulo 4 apresenta o funcionamento da plataforma e o capítulo 5 traz as conclusões e considerações finais.

Capítulo 2

Fundamentação Teórica

2.1 VoIP

Com a infinidade de avanços tecnológicos, como Internet de alta velocidade e o surgimento do *Wi-Fi*, a integração de telefones, computadores, dispositivos móveis como *smartphones* e *tablets* e o surgimento do Skype como um modelo completamente revolucionário para a forma como as pessoas querem se comunicar, o VoIP tornou-se parte do dia-a-dia de muitas pessoas em todo o mundo. A qualidade da chamada foi aprimorada e agora é uma das tecnologias mais faladas no setor de comunicações. Seu crescimento em apenas algumas décadas tem sido surpreendente, e não há como dizer o quanto essa tecnologia ainda pode evoluir.

2.1.1 Origem

A criação do VoIP foi possível utilizando de três grandes invenções do século: o telefone (UOL, 2019), a Internet, que permite o envio e recepção de pacotes de dados e o Protocolo IP.

A empresa VocalTec, em 1995, desenvolveu um dispositivo chamado de Internet Phone, que permitia que o usuário ligasse para outro usuário que possuía um Internet Phone utilizando um *speaker* e um microfone, com a limitação de estarem utilizando o mesmo *software*.

Em 1996, foram desenvolvidos os recursos de correio de voz pela Internet. Assim, foi possível enviar mensagens de voz através da Internet para um telefone convencional, apesar de várias ressalvas como baixa qualidade de som, seguidos períodos de silêncio e quedas de conexão. No mesmo ano, a Vocal Tec anunciou seu *software* em conjunto com o Microsoft NetMeeting, que realiza conferências pela Internet (WIKIPEDIA, 2019).

No final do ano de 1998, surgiram programas de computador, chamados de *softphones*, que eram utilizados para realizar e receber ligações de telefones convencionais. As ligações eram gratuitas, na condição de serem exibidos anúncios ao originador antes e depois da

conversação com o destino.

Os fabricantes de equipamentos telefônicos e especialistas em telecomunicações começaram a usar as recém-desenvolvidas transmissões digitais com o intuito de trafegar pacotes de informações através do Protocolo IP, pelas boas ofertas de velocidade, qualidade e custo. O desenvolvimento de *switches* com recursos de IP e, posteriormente, aparelhos capazes de transmitir o sinal analógico do telefone convencional em sinal digital para trafegar pela internet, chamados de ATAs (Adaptador para Telefone Analógico).

2.1.2 Mercado

Atualmente, o VoIP é difundido em várias aplicações e movimenta anualmente bilhões de dólares. Estima-se que a participação de mercado, em 2015, de ligações internacionais utilizando VoIP foi de 52% da receita total, totalizando aproximadamente U$ 50 bilhões (VOIP-INFO, 2019).

A previsão para os próximos anos é de cada vez maior adoção do VoIP, com a implementação das redes 5G (TRENDS, 2019), que aumentarão significativamente a velocidade de conexão, consequentemente diminuindo perdas de dados e limitações de transmissão, e com a adição de aplicações capazes de interagir com usuários utilizando Inteligência Artificial, como, por exemplo, em automação de atendimento em empresas (SIMKIENE, 2019).

2.1.3 Comunicação utilizando o Protocolo SIP

O SIP (*Session Initiation Protocol*), proposto em 1999 (HANDLEY et al., 1999), possui o funcionamento similar a outro protocolo, utilizado na WEB, o HTTP (*Hypertext Transfer Protocol*) (MOZILLA, 2019b). Baseado no modelo "requisição-resposta", em que a sessão é criada e os participantes devem trocar sinalizações (VOIP-INFO, 2018b), esta tecnologia é utilizada em vários segmentos, como mensagens instantâneas, conferências e principalmente VoIP.

Este protocolo proporciona segurança para troca de dados entre os envolvidos e pode ser utilizado na maioria dos dispositivos de comunicação A Microsoft permitiu que todos os seus dispositivos usassem a tecnologia SIP (MICROSOFT, 2019). O SIP possibilita a comunicação do tipo *Peer-to-Peer* (P2P), ou seja, não depende de servidores ou *gateways*, ou quaisquer outros dispositivos auxiliares.

2.1.4 Definição de termos e conceitos

Ao longo do trabalho, alguns termos e conceitos serão utilizados constantemente, e a fim de padronizar o entendimento do leitor acerca deles, a relação a seguir apresenta aqueles que são essencialmente fundamentais:

- VoIP – Voice over Internet Protocol - Trafegar dados de Voz utilizando a internet, com o protocolo IP.
- PABX IP – Private Automatic Branch Exchange - Sistema que conecta ramais telefônicos, permitindo comunicação interna e externa, sendo esta através de operadoras VoIP, por exemplo.
- Protocolo IAX2 - *Inter Asterisk Exchange* - Protocolo proprietário para comunicação entre sistemas Asterisk.
- *Asterisk* - *Framework* de código aberto utilizado para construir aplicações e sistemas de comunicação.
- *Softswitch* - dispositivo central em uma rede de telecomunicações que conecta chamadas telefônicas de uma linha telefônica a outra, através de uma rede de telecomunicações ou da Internet, inteiramente por meio de *software* rodando em um sistema de computador de rede geral.
- *Softswitch* classe 4 – Softswitch usado para trafegar dados entre operadoras para roteamento de grandes volumes de chamadas de longa distancia utilizando VoIP.
- *Softswitch* classe 5 – São *softswitches* voltados para uso do usuário final, e suas principais características são funções adicionais de PABX IP e outros serviços de callcenter.
- *Codec* – Dispositivo ou *software* utilizado para codificação e decodificação entre sinais analógicos e digitais.
- *Transcoding* – Conversão de sinais de um *codec* para outro *codec*.

2.2 Trabalhos relacionados

No mercado, já existem vários *softwares* utilizados como PABX ou como Plataformas VoIP, e serão apresentados alguns exemplos, destacando as características de cada.

2.2.1 Plataforma MOR

A Plataforma MOR, da empresa Kolmisoft, é uma plataforma VoIP de Classe 5 que está no mercado há anos e possui uma grande quantidade de clientes (KOLMISOFT, 2019). Pode ser utilizada tanto pelo usuário final quanto por um provedor de VoIP com interface gráfica disponível em vários idiomas e, por utilizar Asterisk, permite recebimento de chamadas através de protocolos de comunicação SIP e IAX2 (VOIP-INFO, 2018a).

Em sua documentação, que fornece um vasto material, há instruções para aumentar a capacidade total do serviço com adição de outros servidores de redundância. Além disso,

o sistema é customizável. O serviço, contudo, é pago e sua interface, por se tratar de um projeto desenvolvido há anos, não segue os padrões de design atualmente utilizados, e apesar de possuir suporte para vários idiomas, vários termos técnicos permanecem em inglês e sem explicações breves, prejudicando a usabilidade.

Sua capacidade de chamadas simultâneas é variável, sendo influenciada pelo grau de uso da interface gráfica e pelo uso de *transcoding*. Em um servidor de testes, com um processador de 4 *cores*, o limite é de aproximadamente 100 chamadas simultâneas, sendo necessário uma estrutura robusta e enorme para atender um fluxo considerável de chamadas.

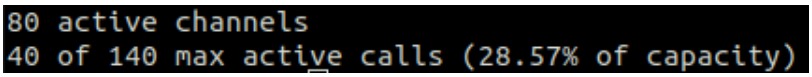

Figura 1 – Estado da Plataforma MOR com 40 chamadas.

2.2.2 Plataforma PyFreeBilling

A Plataforma PyFreeBilling é um projeto *open-source* disponibilizado no GitHub que disponibiliza uma plataforma de chamadas voltada para a operadora VoIP, com sistemas de tarifação e roteamento de chamadas(WOLFF, 2014).

É um projeto recente, e utiliza tecnologias atuais como *Framework* Django para construção da interface do usuário e utiliza o *Framework* FreeSwitch como sistema de comunicação e controle de chamadas. Possui um painel de controle do sistema completo e robusto, instruções e permite customização tanto a nível de interface gráfica quanto de sistema de comunicação.

Suporta em torno de 1000 chamadas, conforme dito pelo criador, porém não foi especificado os requisitos mínimos de sistema, e escalabilidade do sistema não foi especificada. Houve relatos de problemas de segurança que permitiram completamento indevido de chamadas por terceiros em contas que possuíam autenticação de usuário por IP, porém as contas com autenticação por usuário e senha não apresentaram problema.

A interface gráfica, apesar de completa, somente suporta o idioma inglês e possui algumas funções muito confusas e desorganizadas entre os menus.

Capítulo 3

Projeto Praha

A Praha é o projeto de *backend* de plataforma VoIP proposto por este trabalho que as funções básicas de controle de usuários, configurações de roteamento e tarifação de chamadas, utilizando tecnologias atuais, arquitetura RESTful e disponibilizando APIs, e possui alta capacidade de chamadas por servidor. A Praha pode ser utilizada, por exemplo, diretamente no cliente, para simplificar a conexão dos servidores com seus provedores. A capacidade de chamadas simultâneas gira em torno por volta de 450[1], sistema completo de tarifação para provedores e usuários, painel de monitoramento de chamadas e recursos do servidor.

3.1 Comparação entre a Praha e outras plataformas

A Plataforma Praha oferece simplicidade e alta customização do sistema. Quando comparada com os projetos citados, como a MOR e a PyFreeBilling, a Praha não oferece tantas funcionalidades ou cenários de uso, porém fornece um sistema de telefonia funcional, estável e que possua as funções mínimas para realizar o roteamento de chamadas, gerenciamento de custos e monitoramento de servidor, capaz de lidar com uma grande capacidade de chamadas simultâneas, podendo ser utilizado em diferentes cenários, como servidor central de telefonia de uma empresa ou de um provedor VoIP.

[1] Teste efetuado em servidor com processador de 4 *cores*

Tabela 1 – Comparação entre Plataformas

Categoria	**Plataforma Mor**	**PyFreeBilling**	**Praha**
Escalabilidade	Possível	Não Especificado	Não Especificado
Usabilidade	Ruim	Ruim	Não Aplicável
Chamadas por Servidor	100*	1000**	450*
Segurança	Bom	Ruim	Bom
Customização	Ruim	Bom	Não Aplicável
Suporte a Idiomas	Excelente	Ruim	Não Aplicável
Classe	5	4	4
Valor	Pago	Gratuito	Pago

* Testes efetuados em um servidor com as mesmas configurações

** Quantidade determinada pelo criador, não especificando os requisitos mínimos de *hardware*.

3.2 Modelagem

Um modelo é um conjunto de asserções sobre a estrutura ou comportamento de um sistema pelo qual procura-se explicar ou prever as suas propriedades do sistema. Em outras palavras, modelos são, em sua maioria, abstrações simplificadas da estrutura e funcionamento de sistemas reais. Através de um processo de modelagem são realizados os devidos refinamentos para a definição e construção de um modelo.

Para construir a Praha foi necessário incluir uma etapa de modelagem, sobretudo dos componentes que a compõe. A modelagem foi feita em UML, uma vez que ela permite modelar tanto sistemas comerciais simples quanto sistemas de alto desempenho, como é o caso deste projeto.

3.3 Desenvolvimento

Primeiramente, a Praha foi concebida em componentes, suas interconexões e principais serviços ofertados. Somente aspectos relevantes foram incluídos, pois muitos detalhes podem interferir no processo de representação do comportamento desejado e, além disso, não gera vantagens reais que justifique seu custo de elaboração. A Figura 2 mostra o diagrama de componentes elaborado como elemento de alto nível de abstração da Praha.

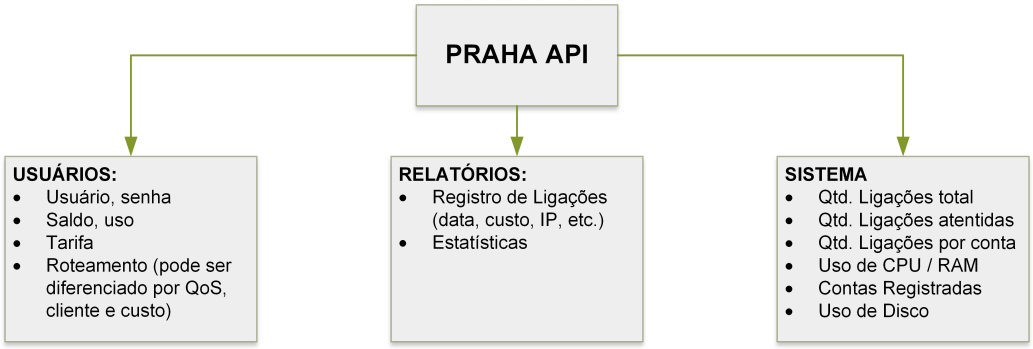

Figura 2 – Diagrama de Componentes da Plataforma Praha

3.3.1 Tecnologias utilizadas

Para o desenvolvimento do projeto, foram utilizadas tecnologias *open-source* atuais, robustas, seguras e bastante comuns no mercado.

3.3.1.1 Python

A linguagem Python (PYTHON.ORG, 2019) é uma excelente linguagem de programação, devido à sua sintaxe de fácil compreensão, capacidade de programação em alto nível e é uma das linguagens mais requisitadas atualmente (SAEED, 2018).

3.3.1.2 Django

Django é um framework para desenvolvimento WEB (PROJECT, 2019) que fornece ferramentas robustas para utilização na criação de sistemas e foi escolhido pela praticidade, segurança e facilidade na criação de aplicações com arquitetura RESTful. É utilizado em grandes aplicações, como Instagram e BitBucket.

3.3.1.3 FreeSwitch

FreeSwitch é um sistema de telefonia robusto e *open-source* e foi alta performance e facilidade de implementação.

3.3.1.4 MySQL

MySQL é um sistema gerenciador de Banco de Dados robusto e bastante utilizado comercialmente. Foi escolhido por oferecer uma excelente performance e por experiência com o *software*.

3.3.2 Módulos

A Praha possui 4 módulos primários, sendo eles Usuários, Sistema, Telefonia e Relatórios. São responsáveis por controlar a plataforma e retornar informações através de APIs.

3.3.2.1 Sistema

O módulo Sistema fornece APIs de verificação do estado do servidor, em tempo real, e por controlar a escrita dos arquivos de configuração

- ❏ Quantidade de ligações totais e atendidas
- ❏ Quantidade de contas VoIP registradas
- ❏ Uso de Disco separado por partição
- ❏ Uso de CPU e Memória RAM

3.3.2.2 Relatórios

O módulo Relatórios fornece APIs para consulta e criação de estatísticas de uso através dos registros de ligações, com informações de usuário, número de origem, número de destino, tempo de ligação, entre outros.

- ❏ Registro de ligações
- ❏ Exportar em formato CSV

3.3.2.3 Usuários

O módulo Usuários fornece APIs para gerenciamento de usuários cadastrados, com todas as operações básicas de banco de dados relacionais: listagem, criação, atualização, e remoção. Cada usuário possui uma regra de roteamento e uma tarifa vinculada, que são consultadas a cada requisição de ligação.

- ❏ Criação, Listagem, Atualização e Remoção de Usuários

3.3.2.4 Telefonia

O módulo Telefonia fornece APIs para gerenciamento de contas VoIP, troncos/provedores, regras de roteamento e tarifas, com todas as operações básicas de banco de dados relacionais (Figura 3). Também é responsável pela escrita dos arquivos de configuração do sistema.

- ❏ Criação, Listagem, Atualização e Remoção de Contas VoIP

- ❏ Criação, Listagem, Atualização e Remoção de Troncos/Provedores
- ❏ Criação, Listagem, Atualização e Remoção de Roteamentos
- ❏ Criação, Listagem, Atualização e Remoção de Tarifas
- ❏ Manipulação dos arquivos de configuração do sistema

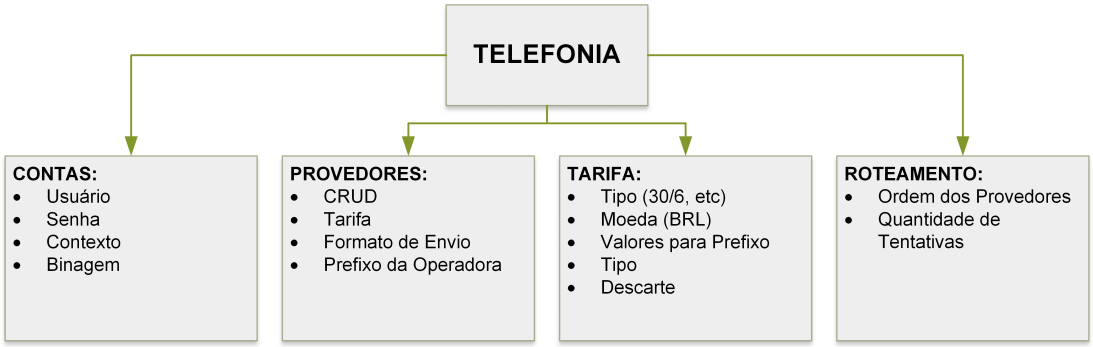

Figura 3 – Detalhamento do módulo de Telefonia

Capítulo 4

Funcionamento da Plataforma

O *software* de comunicação FreeSwitch, base da Plataforma, possui suas configurações estruturadas em arquivos de formato XML, desde a informações de contas VoIP (4), cadastro de troncos (operadoras VoIP) e regras de discagem (sequência de instruções a serem executadas a cada requisição de ligação).

```
<include>
    <user id="1000">
        <params>
            <param name="password" value="gIVEypu0Wk" />
        </params>
        <variables>
            <variable name="user_context" value="default" />
            <variable name="callgroup" value="1" />
        </variables>
    </user>
</include>
```

Figura 4 – Arquivo XML de configuração de uma conta VoIP

A Figura 5 mostra um exemplo de arquivo XML de configuração de um Tronco (Operadora) VoIP.

```
<include>
    <gateway name="trunk-for-testing">
        <param name="username" value="1212" />
        <param name="password" value="123456" />
        <param name="realm" value="123.124.221.255:5060" />
        <param name="register" value="true" />
        <param name="from-user" value="1212" />
        <param name="from-domain" value="123.124.221.255" />
    </gateway>
</include>
```

Figura 5 – Arquivo XML de configuração de um Tronco.

4.1 Ciclo de uma Chamada

Quando uma chamada é recebida pela plataforma, através de um usuário devidamente autenticado, é executado o código A.1 que realiza a verificação e consulta das informações necessárias para prosseguir com a chamada. Caso ocorra erro em alguma etapa, ou a verificação não permitiu o prosseguimento da chamada, como, por exemplo, usuário inativo, a chamada é desconectada. São seguidos os seguintes passos:

1. Consultar os dados da conta VoIP.
2. Consultar o usuário vinculado à conta. Caso inativo, desconecta a chamada
3. Verifica se o usuário possui uma Tarifa vinculada. Caso não possua, desconecta a chamada.
4. Consulta o tipo de tarifação do usuário.
5. Consulta o valor do minuto da ligação para o destino.
6. Verifica se o usuário pode realizar a ligação com base no valor de 1 minuto de ligação. Se não puder, desconecta a chamada.
7. Consulta qual a regra de roteamento do usuário.
8. Consulta quais troncos/provedores pertencem a esse roteamento e a ordem deles. Caso não encontre nenhum, desconecta a chamada.
9. Consulta quais as tarifas de cada tronco/provedor.
10. Termina a execução do código para seguir com as configurações de roteamento, ou seja, enviar a chamada para os devidos troncos/provedores.

Após a ligação ser desconectada, são executados os códigos A.1 e A.3, para que sejam processados os custos e as informações necessárias para o registro da ligação.

4.2 Largura de Banda

Cada ligação utiliza um tamanho de banda para trafegar os dados pela Internet. O *codec* G711, que não realiza compressão dos dados e mantém a qualidade da voz, utiliza até 64 Kbps de banda por ligação, e o *codec* G729, que realiza compressão dos dados e sacrifica parte da qualidade, utiliza até 30 Kbps de banda por ligação (CISCO, 2019).

Outros *codecs*, assim como o G729, podem utilizar até 1/3 da banda por comprimir os dados, com menor qualidade de áudio, e é mais utilizado para VoIP, onde essa troca é válida, já que consegue manter uma conversação com qualidade aceitável. Na Plataforma Praha, são utilizados os *codecs* G729 e G711.

4.3 Testes

Foram efetuados testes de ligação por dois meios: uma aplicação para simular uma situação real de um usuário conectando a outro e um código que efetua várias requisições, a fim de analisar a capacidade de chamadas simultâneas.

4.3.1 Zoiper

O *software* Zoiper (ZOIPER, 2019) é um *softphone* que permite realizar ligações através do Protocolo SIP. As ligações unitárias foram realizadas pela aplicação, validando a sinalização e reprodução de áudio entre as partes conectadas.

4.3.2 Código

O código A.4 foi feito para realizar um determinado número de chamadas, de forma que a plataforma processe cada requisição e as tratando como uma tentativa válida, ou seja, são efetuados todos os procedimentos de uma chamada legal. As requisições provenientes deste código ficam em espera durante 10 segundos, para garantir que todas as chamadas estarão ativas concorrentemente.

Cada chamada cria duas sessões, sendo uma para conexão do usuário com o servidor e outra para conexão do servidor com o provedor. No teste, foram geradas 450 chamadas, que equivalem a 900 sessões ativas.

4.4 Problemas Conhecidos da Praha

Ambos os problemas conhecidos da Praha são referentes a segurança, e formas de ataques que eventualmente podem comprometer sua integridade. São eles:

Ataque de Tentativa de Registro: múltiplas tentativas de registrar em uma conta tentando adivinhar a senha. Normalmente há softwares (ex: Fail2Ban) que analisam e bloqueiam os IP's depois de um certo número de tentativas, porém quando a quantidade de tentativas é muito grande, o software não consegue agir em 100% dos casos. Em cenário prático, esses softwares são bem efetivos.

IP Spoofing: caso a plataforma aceite receber chamadas apenas autenticando por IP, o atacante pode conseguir "mascarar" seu IP para tentar realizar chamadas. Em cenário prático, não é comum acontecer este tipo de situação.

Figura 6 – Zoiper, aplicação para realizar chamadas por VoIP.

```
FreeSWITCH (Version 1.6.20  64bit) is ready
0 session(s) since startup
0 session(s) - peak 0, last 5min 0
0 session(s) per Sec out of max 200, peak 0, last 5min 0
1000 session(s) max
min idle cpu 0.00/99.20
Current Stack Size/Max 240K/8192K
```

Figura 7 – Estado do Sistema com 0 chamadas.

```
FreeSWITCH (Version 1.6.20  64bit) is ready
900 session(s) since startup
900 session(s) - peak 900, last 5min 900
130 session(s) per Sec out of max 200, peak 177, last 5min 177
1000 session(s) max
min idle cpu 0.00/81.47
Current Stack Size/Max 240K/8192K
```

Figura 8 – Estado do Sistema com 450 chamadas simultâneas.

Capítulo 5

Conclusão

O objetivo deste trabalho foi desenvolver o *backend* de uma plataforma VoIP, utilizando a arquitetura RESTful para oferecer APIs, capaz de lidar com uma grande quantidade de chamadas, especificando o módulo responsável por cuidar do roteamento de uma chamada e a modelagem dos dados utilizados pelo *software*.

Ao final deste trabalho, diante dos resultados apresentados, pode-se concluir que o objetivo proposto para este trabalho de conclusão de curso foi atingido, e, mais que isso, ficou claro que a viabilidade deste tipo de iniciativa revela inúmeras possibilidades de criação de aplicações que utilizem a abordagem adotada neste trabalho.

5.1 Trabalhos Futuros

Para trabalhos futuros, pode-se implementar formas de escalar o serviço, utilizando, por exemplo, um software capaz de realizar a distribuição de chamadas para várias instâncias da plataforma. Alguns exemplos de softwares desta natureza são (OPENSIPS, 2018) e (KAMAILIO, 2018).

Também pode-se permitir outro método de recebimento de chamadas, definindo um TechPrefix (código único a ser enviado juntamente com o número discado, como, por exemplo, 980# + NÚMERO) e liberando o IP do usuário. Esta função é importante em cenários comerciais onde os servidores dos clientes podem perder a autenticação com a plataforma periodicamente.

E por último, pode-se implementar um software utilizando tecnologias de FrontEnd para consumir os recursos providos pela arquitetura de API da Plataforma Praha.

Apêndices

APÊNDICE **A**

Trechos de códigos

Os trechos de código a seguir, escritos na linguagem de programação Python, são utilizados ao realizar e ao finalizar uma ligação na Plataforma Praha.

A.1 Roteamento de Chamadas

Quando uma chamada é recebida pela Plataforma, este script é executado para verificar se pode prosseguir, definindo algumas variáveis de ligação como usuário, custo do minuto da ligação, entre outras informações necessárias para tarifação após término da chamada.

call_routing.py

```
# Script responsible for receiving call request and set routing configuration

import pymysql
from .db_params import DB_HOST, DB_USER, DB_PASSWORD, DB_PORT, DB_NAME


def handler(session, args):
    connection = pymysql.connect(host=DB_HOST, user=DB_USER, password=DB_PASSWORD, port=DB_PORT, db=DB_NAME,
                                 charset='utf8mb4', cursorclass=pymysql.cursors.DictCursor)
    args = args.split(' ')
    args = {k: v for k, v in list(map(lambda x: x.split('='), args))}
    username = str(args['username'])
    destination_number = str(args['destination_number'])
    call_params = {}
    with connection.cursor() as cursor:
        query = 'SELECT id, username, user_id, caller_id_number FROM telephony_peer WHERE username=%s'
        cursor.execute(query, [username])
        peer = cursor.fetchone()
    if not peer:
        connection.close()
        session.execute('log', 'MESSAGE: PEER {} NOT FOUND'.format(username))
        session.execute('hangup')
        exit()
    with connection.cursor() as cursor:
        query = 'SELECT id,username,tariff_id,is_active,balance,threshold,routing_id FROM accounts_user WHERE id=%s'
        cursor.execute(query, [peer['user_id']])
        user = cursor.fetchone()
    if user:
        if not user['is_active']:
            connection.close()
            session.execute('log', 'MESSAGE: USER IS NOT ACTIVE')
            session.execute('hangup')
            exit()
    else:
        connection.close()
        session.execute('log', 'MESSAGE: USER NOT FOUND')
        session.execute('hangup')
        exit()
    if not user['tariff_id']:
        connection.close()
        session.execute('log', 'MESSAGE: USER DOES NOT HAVE TARIFF')
        session.execute('hangup')
        exit()
    phone_number = destination_number
    with connection.cursor() as cursor:
        query = 'SELECT HIGH_PRIORITY billing_round FROM telephony_tariff WHERE id=%s'
        cursor.execute(query, [str(user['tariff_id'])])
        tariff_info = cursor.fetchone()
    if not tariff_info:
        connection.close()
        session.execute('log', 'MESSAGE: TARIFF NOT FOUND')
        session.execute('hangup')
        exit()
    rate = None
    while True:
        with connection.cursor() as cursor:
            query = 'SELECT HIGH_PRIORITY value,discard,rate_type FROM telephony_rate WHERE tariff_id=%s AND prefix=%s'
            length = cursor.execute(query, [user['tariff_id'], phone_number])
            if length > 1:
                break
            rate = cursor.fetchone()
            if not rate:
                phone_number = phone_number[0:-1]
            if phone_number == '' or rate is not None:
                break
    if not rate:
        connection.close()
        session.execute('log', 'MESSAGE: USER RATE NOT FOUND')
        session.execute('hangup')
        exit()
    if length > 1:
        connection.close()
        session.execute('log', 'MESSAGE: TOO MANY CUSTOMER RATES FOUND')
        session.execute('hangup')
        exit()
    tariff = {
        'billing_round': tariff_info['billing_round'],
        'value': rate['value'],
        'discard': rate['discard'],
        'type': rate['rate_type']
    }
    if (user['balance'] - tariff['value']) < user['threshold']:
        connection.close()
        session.execute('log', 'MESSAGE: USER THRESHOLD REACHED.')
        session.execute('hangup')
        exit()
    call_params['user_id'] = str(user['id'])
    call_params['user_username'] = str(user['username'])
```

```
call_params['call_type'] = str(tariff['type'])
call_params['customer_billing_round'] = str(tariff['billing_round'])
call_params['customer_rate'] = str(tariff['value'])
call_params['customer_discard'] = str(tariff['discard'])
with connection.cursor() as cursor:
    query = 'SELECT name, slug, active FROM telephony_routing WHERE id=%s'
    cursor.execute(query, [str(user['routing_id'])])
    routing_info = cursor.fetchone()
if routing_info:
    if not routing_info['active']:
        connection.close()
        session.execute('log', 'MESSAGE: LCR INACTIVE')
        session.execute('hangup')
        exit()
else:
    connection.close()
    session.execute('log', 'MESSAGE: LCR NOT FOUND')
    session.execute('hangup')
    exit()
with connection.cursor() as cursor:
    query = 'SELECT number, trunk_id FROM telephony_priority WHERE routing_id=%s'
    cursor.execute(query, [str(user['routing_id'])])
    routing_priorities = cursor.fetchall()
if not routing_priorities:
    connection.close()
    session.execute('log', 'MESSAGE: ROUTING {} EMPTY'.format(routing_info['name']))
    session.execute('hangup')
    exit()
for priority in routing_priorities:
    with connection.cursor() as cursor:
        query = 'SELECT name, tariff_id FROM telephony_trunk WHERE id=%s'
        cursor.execute(query, [str(priority['trunk_id'])])
        trunk_info = cursor.fetchone()
    if not trunk_info:
        connection.close()
        session.execute('log', 'MESSAGE: COULD NOT RETRIEVE TRUNK ID {}'.format(str(priority['trunk_id'])))
        session.execute('hangup')
        exit()
    with connection.cursor() as cursor:
        query = 'SELECT billing_round, currency_rate FROM telephony_tariff WHERE id=%s'
        cursor.execute(query, [str(trunk_info['tariff_id'])])
        tariff_info = cursor.fetchone()
    if not tariff_info:
        connection.close()
        session.execute('log', 'MESSAGE: COULD NOT RETRIEVE TARIFF ID {}'.format(str(trunk_info['tariff_id'])))
        session.execute('hangup')
        exit()
    phone_number = destination_number
    rate_info = None
    while True:
        with connection.cursor() as cursor:
            query = 'SELECT value, discard FROM telephony_rate WHERE prefix=%s AND tariff_id=%s'
            length = cursor.execute(query, [str(phone_number), str(trunk_info['tariff_id'])])
        if length > 1:
            break
        rate_info = cursor.fetchone()
        if not rate_info:
            phone_number = phone_number[0:-1]
        if phone_number == '' or rate_info is not None:
            break
    if not rate_info:
        connection.close()
        session.execute('log', 'MESSAGE: TRUNK RATE NOT FOUND')
        session.execute('hangup')
        exit()
    if length > 1:
        connection.close()
        session.execute('log', 'MESSAGE: TOO MANY TRUNK RATES FOUND')
        session.execute('hangup')
        exit()
    call_params['trunk_name_{}'.format(str(priority['number']))] = str(trunk_info['name'])
    call_params['trunk_rate_{}'.format(str(priority['number']))] = str(rate_info['value'])
    if tariff_info['currency_rate']:
        if tariff_info['currency_rate'] >= 0:
            call_params['trunk_rate_{}'.format(str(priority['number']))] *= tariff_info['currency_rate']
    call_params['trunk_discard_{}'.format(str(priority['number']))] = str(rate_info['discard'])
    call_params['trunk_billing_round_{}'.format(str(priority['number']))] = str(tariff_info['billing_round'])
connection.close()
call_params['user_routing'] = str(routing_info['slug'])
if peer['caller_id_number']:
    call_params['effective_caller_id_name'] = str(peer['caller_id_number'])
    call_params['effective_caller_id_number'] = str(peer['caller_id_number'])
session.execute('multiset', ' '.join(['{}={}'.format(key,value) for key, value in call_params.items()]))
session.transfer('${user_routing}', 'XML', 'default')
return
```

A.2 Tarifação da Chamada

Após a ligação ser finalizada, são feitos os cálculos de custo do cliente e do fornecedor, para ser posteriormente verificado no registro da ligação.

call_billing.py

```python
# Perform call billing based on customer and trunk costs

import math
import pymysql
from .db_params import DB_HOST, DB_USER, DB_PASSWORD, DB_PORT, DB_NAME


def fsapi(session, stream, env, args):
    billsec = int(session.getVariable('billusec'))
    billsec = int(math.ceil(billsec / float(1000000)))
    if billsec == 0:
        stream.write('0')
        return
    user_id = session.getVariable('user_id')
    if args == 'client':
        user_id = session.getVariable('user_id')
        billing_round = session.getVariable('customer_billing_round')
        rate = float(session.getVariable('customer_rate'))
        discard = int(session.getVariable('customer_discard'))
    else:
        billing_round = session.getVariable('trunk_billing_round')
        rate = float(session.getVariable('trunk_rate'))
        discard = int(session.getVariable('trunk_discard'))
    cost = calc_value(billing_round, rate, billsec, discard)
    if cost > 0 and args == 'client':
        connection = pymysql.connect(host=DB_HOST, user=DB_USER, password=DB_PASSWORD, port=DB_PORT, db=DB_NAME,
                                     charset='utf8mb4', cursorclass=pymysql.cursors.DictCursor)
        with connection.cursor() as cursor:
            query = 'UPDATE accounts_user SET balance = balance-%s WHERE id=%s'
            cursor.execute(query, (str(cost), user_id))
        connection.commit()
        connection.close()
    stream.write(str(cost))


def calc_value(billing_round, rate, billsec, discard=0):
    rate = float(rate)
    billsec = int(billsec)
    discard = int(discard)
    cost = 0.0
    if billsec <= discard:
        return cost
    if billing_round == '30/6':
        if billsec <= 30:
            cost = rate / 2
        else:
            billsec -= 30
            cost = (rate / 2) + (math.ceil(float(billsec) / 6) * ((rate / 2) / 5))
    elif billing_round == '1/1':
        cost = (float(rate) / 60) * billsec
    else: # billing_round == '6/6'
        cost = math.ceil(float(billsec) / 6) * (rate / 10)
    return round(cost, 6)
```

A.3 Arredondar o tempo falado

A variável que indica o tempo falado da ligação está na ordem de microsegundos. Este script é executado após a ligação ser finalizada e faz a conversão para segundos e arredonda para a próxima casa decimal.

round_billsec.py

```python
# Round billing seconds before performing actual call billing

import math


def fsapi(session, stream, env, args):
    # For a more accurate number of billing seconds, get variable billusec
    billsec = int(session.getVariable('billusec'))
    # This variable returns as microseconds, so it has to be converted to seconds
    # dividing it by 1 million and rounding it up to next integer number.
    # EX: 1,5 seconds becomes 2 seconds, and so on.
    billsec = int(math.ceil(billsec/float(1000000)))
    stream.write(str(billsec))
    return
```

A.4 Originar Chamadas

Origina uma definida quantidade de chamadas para simular um cenário de produção, onde o sistema recebe várias ligações por segundo.

originate_calls.py

```
from billing.telephony.utils import FreeSwitch
from threading import Thread


class FSThread(Thread):

    def run(self):
        freeswitch = FreeSwitch()
        freeswitch.connect()
        if freeswitch.connected:
            freeswitch.execute('originate sofia/gateway/trunk/5511999887766 test XML default')
        freeswitch.disconnect()
        return


CALLS = 450

for i in range(CALLS):
    fs_thread = FSThread()
    fs_thread.start()
```

Referências

CISCO. Voice over ip - per call bandwidth consumption. **https://www.cisco.com/c/en/us/support/docs/voice/voice-quality/7934-bwidth-consume.htmlanc3**, 2019. Acesso em: 12.07.2019.

HANDLEY, M. et al. Sip: Session initiation protocol. **tools.ietf.org/html/rfc2543**, 1999. Acesso em: 13.07.2019.

KAMAILIO. Kamailio - an open source sip server. **kamailio.org/**, 2018. Acesso em: 24.10.2011.

KOLMISOFT. Mor – class 5 softswitch with billing and routing. **VoIP-Info.org**, 2019. Acesso em: 11.07.2019.

MICROSOFT. [ms-sip]: Session initiation protocol extensions. **docs.microsoft.com/en-us/openspecs/windows**$_p rotocols/ms - sip/0d72fb55 - 12ba - 49f1 - 850d - f0bf3110bb1f, 2019. Acessoem$: 14.07.2019.

MOZILLA. Rest. **https://developer.mozilla.org/en-US/docs/Glossary/REST**, 2019. Acesso em: 17.07.2019.

_____. Uma visão geral do http. **https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Overview**, 2019. Acesso em: 13.07.2019.

OPENSIPS. Opensips - an open source sip proxy/server for voice, video, im, presence and any other sip extensions. **opensips.org/**, 2018. Acesso em: 24.10.2011.

PROJECT, D. Django, the web framework for perfectionists with deadlines. **www.djangoproject.com/**, 2019.

PYTHON.ORG. Python programming language. **https://www.python.org/**, 2019.

SAEED, A. Here are the ten best programming languages to learn in 2019. **codinginfinite.com/best-programming-languages-to-learn-2019/**, 2018. Acesso em: 14.07.2019.

SIMKIENE, V. Predictions for voip services and technologies in 2019. **voip.review/2019/02/20/predictions-voip-services-technologies-2019/**, 2019. Acesso em: 14.07.2019.

TRENDS, D. Everything you need to know about 5g. **https://www.digitaltrends.com/mobile/what-is-5g/**, 2019. Acesso em: 14.07.2019.

UOL. Alexander graham bell. **educacao.uol.com.br/biografias/alexander-graham-bell.htm**, 2019. Acesso em: 13.07.2019.

VOIP-INFO. Iax – inter asterisk exchange. **VoIP-Info.org**, 2018. Acesso em: 24.10.2011.

_____. Sip – session initiation protocol. **www.voip-info.org/sip/**, 2018. Acesso em: 24.10.2011.

_____. What is voip. **https://www.voip-info.org/what-is-voip/**, 2019. Acesso em: 13.07.2019.

WIKIPEDIA. Netmeeting. **pt.wikipedia.org/wiki/NetMeeting**, 2019. Acesso em: 13.07.2019.

WOLFF, M. Pyfreebilling – open source voip billing and routing platform. **PyFreeBilling.com**, 2014. Acesso em: 24.10.2011.

ZOIPER. Zoiper softphone. **https://www.zoiper.com/**, 2019.