Renato Müller Rosa

Sistema de Tarifação para FreeSWITCH

São José – SC

Agosto / 2015

Renato Müller Rosa

Sistema de Tarifação para FreeSWITCH

Monografia apresentada à Coordenação do Curso Superior de Tecnologia em Sistemas de Telecomunicações do Instituto Federal de Santa Catarina para a obtenção do diploma de Tecnólogo em Sistemas de Telecomunicações.

Orientador:
Prof. Marcelo Maia Sobral, Dr. Eng.

CURSO SUPERIOR DE TECNOLOGIA EM SISTEMAS DE TELECOMUNICAÇÕES
INSTITUTO FEDERAL DE SANTA CATARINA

São José – SC

Agosto / 2015

Monografia sob o título *"Sistema de Tarifação para FreeSWITCH"*, defendida por Renato Müller Rosa e aprovada em 20 de Agosto de 2015, em São José, Santa Catarina, pela banca examinadora assim constituída:

Prof. Marcelo Maia Sobral, Dr. Eng.
Orientador

Prof. Ederson Torresini, M.Sc.
IFSC

Prof. Tulio Alberton Ribeiro, M.Sc.
IFSC

Eu nunca conheci um gênio.
Um gênio para mim é alguém que faz bem algo que ele odeia.
Qualquer um pode fazer bem algo que ele ama,
é apenas uma questão de encontrar o assunto.
Clint Eastwood

Agradecimentos

Gostaria de agradecer inicialmente ao Instituto Federal de Santa Catarina, campus São José, por ter me dado a oportunidade de fazer mais um curso em suas instalações. Ao professor Marcelo Maia Sobral, pela dedicação, paciência e ter se mostrado sempre disposto a aceitar novamente ser meu orientador, mesmo demorando todo o tempo do mundo para desenvolver e escrever este trabalho.

Gostaria de agradecer também aos meus pais, por terem aceitado durante tanto tempo a desculpa de que estava fazendo meu trabalho. Também ao meu amigo Carlos Eduardo Wagner, por ter me dado a ideia do tema do trabalho e ter me ajudado com vários pontos cruciais do trabalho. E a todos aos amigos que conheci durante o curso, principalmente ao Rafael da Silva Pereira, por ser o grande amigo que se tornou hoje e o João Carlos Warmling por estar sempre me cobrando quando vou me formar.

Resumo

Atualmente a telefonia, como um todo, é algo comum para a maioria da população e mesmo sem as pessoas verem, o número de pessoas que passam a utilizar telefonia IP cresce cada vez mais. A proposta desse trabalho é acompanhar esse crescimento e com isso disponibilizar mais uma ferramenta que pode ser utilizada para a tarifação telefônica pré-paga e pós-paga. Para isso foi desenvolvido um sistema de tarifação que conta com um tarifador pré-pago em tempo real e também pós-pago, juntamente de uma interface administrativa para facilitar o cadastro de dados, como os planos, planos de discagem, assinantes e tarifas. Essa ferramenta se utiliza de módulos disponibilizados juntamente do FreeSWITCH e da plataforma Java para o seu desenvolvimento.

Abstract

Nowadays telecommunication as a whole is something ordinary for most of the population and even though most people don't see it, the number of people that is using IP telecommunications is growing more and more. The purpose of this work is to follow this growth and to provide a tool that can be used for pre-paid and post-paid telephone billing. For that, a billing system with both pre-paid and post-paid real time billing, along with a managing interface that eases data registering such as plans, dialing plans, users and bills, was developed. This tool uses modules provided along with the FreeSWITCH and the Java platform for it's development.

Sumário

Lista de Figuras

1 Introdução

Com o crescimento do *Voice over Internet Protocol* (VoIP) nos últimos tempos, a necessidade de novas aplicações e que façam mais do que encaminhar chamadas, como gerar relatórios, mostrar custos, se faz mais presente. Há tempos existem aplicações baseadas em Asterisk, um PABX IP muito conhecido no mercado de telefonia IP, como Elastix, Trixbox e FreePBX que servem para controle e configuração do mesmo para centrais telefônicas IP (*Internet Protocol*) de pequeno e médio porte. E para ajudar na tarifação existe o *a2billing* (Asterisk2billing), que calcula custos de ligações a partir de registros de chamadas feitos pelo Asterisk, além de gerar estatísticas e relatórios baseado em uma série de informações configuradas pelo próprio usuário. Essa tarifação é vital para operadoras telefônicas, e útil para empresas que procuram ter uma fonte aproximada de quanto estão gastando.

Com a necessidade de uma aplicação telefônica com capacidade de intermediar uma quantidade maior de chamadas simultâneas e que fosse menos complexo de configurar que o OpenSIPS, que é um SIP proxy[1] compatível com *Session Initiation Protocol* (SIP), destinado para aplicações de larga escala (GONCALVES, 2010), foi criado o FreeSWITCH. Diferente do Asterisk, o FreeSWITCH é um *softswitch*, que é um *software* que conecta ligações de uma linha telefônica a outra através de um *software*. O Asterisk tem implementado em seu núcleo um *softswitch*, porém, suas funcionalidades foram pensadas para a implementação de um PABX, diferente do FreeSWITCH que acaba se tornando mais versátil por ser apenas um *softswitch*. O FreeSWITCH foi feito para encaminhar chamadas, podendo suportar milhares de chamadas simultâneas, e é escalável (MINESSALE, 2013). Porém ele é um *software* considerado novo, já que amadurecimento de um *software* em telefonia é algo lento. Foi criado em 2006, e tem uma proposta diferente do Asterisk e muito menos tempo e reconhecimento no mercado. Hoje é complicado achar aplicações iguais as que temos baseadas em Asterisk, como por exemplo o já citado *a2billing*.

Visando a utilização do FreeSWITCH como um PABX IP, este trabalho propõe o desenvolvimento de um sistema de tarifação de chamadas para o FreeSWITCH. Um sitema de tarifação

[1] Responsável por registrar os usuários e manter a localização desses usuários (GONCALVES, 2010)

de chamadas nada mais é do que um sistema que vai adicionar valores monetários em cima de tempo de chamadas. Hoje existem alguns sistemas de tarifação para FreeSWITCH tanto de código aberto como o ASTPP e vBilling e alguns proprietários que são para diferentes plataformas e não apenas FreeSWITCH. O sistema de tarifação desse trabalho pretende implementar uma tarifação pré e pós paga para o FreeSWITCH, assim como fazer o controle de assinantes, planos de discagem e tarifas. Com isso, o tarifador poderá calcular custos de ligações em cima dos bilhetes gerados pelo FreeSWITCH.

1.1 Objetivo do trabalho

O objetivo principal desse trabalho é desenvolver um sistema de tarifação para as chamadas saintes do FreeSWITCH, tanto de forma pré-paga, quanto pós-paga. Busca-se ainda:

- Desenvolver interfaces para cadastro de assinantes, planos de discagem, tarifas e adicionar créditos a um assinante;
- Gerar arquivo para a exportação da tarifação pós paga;

1.2 Organização do texto

No capítulo 2 são descritas as tecnologias utilizadas para o desenvolvimento do tarifador, assim como um breve entendimento do que seria um tarifador. No capítulo 3 é mostrado como o sistema foi desenvolvido, suas telas para cadastro de dados e como funciona a tarifação do sistema. Por fim, no capítulo 4, são apresentadas observações sobre o funcionamento geral do sistema projetado.

2 *Fundamentação Teórica*

2.1 Tarifação

Segundo Edoardo Boechat (BOECHAT, 2005), um sistema de tarifação deve ser capaz de precificar qualquer tipo de produto, serviço ou ambos. A partir desta precificação, o sistema deve gerar faturas de cobrança para serem enviadas aos clientes. Podemos classificar como produto todos os bens físicos de consumo cuja unidade de medida seja o número de "peças"consumidas. No caso da telefonia o produto é o tráfego de voz, e a forma utilizada atualmente para precificar esse produto é sua duração em minutos.

Para o sistema de tarifação funcionar corretamente, é necessário que o sistema colete as informações de consumo das chamadas, que é o tempo que essa chamada durou. Essas informações são armazenadas em registros chamados de CDR (*Call Detail Record*) 2.1.1. Esses registros servem tanto para a tarifação, como também pode ser utilizado para geração de relatórios gerenciais. Eles contem detalhes das chamadas, como a hora exata que a chamada se iniciou, qual número iniciou a chamada, para que número foi encaminhada a chamada, o tempo total, qual hora terminou, entre outros dados.

Com os dados do CDR, é possível tarifar e atribuir valores a serem cobrados sobre o serviço de voz. Essa parte pode sofrer influência de algumas variáveis. Por exemplo, na telefonia brasileira existe o DDD (Discagem Direta a Distância) e DDI (Discagem Direta Internacional). Então caso uma chamada for de algum dos dois tipos citados anteriormente, ela tem tarifação de forma diferenciada, com valores diferenciados dependendo da distância entre a pessoa que está ligando e a pessoa que está recebendo a chamada. Também existem tarifação diferenciada dependendo do horário, por exemplo a operadora telefônica Oi tem em seu site[1] dois horários definidos, um como Simples e outro Normal. O Horário Simples é cobrado um valor único por chamada, que é correspondente ao valor de dois minutos, já o Horário Normal é cobrado o valor equivalente a 30 segundos quando a chamada for atendida e após isso é cobrado a cada 6

[1] http://www.oi.com.br/oi/oi-pra-voce/planos-servicos/oi-fixo/planos/plano-basico-de-minutos

segundos.

2.1.1 CDR

Centrais telefônicas geram o chamado *Call Detail Record*, traduzido para Registro de Detalhes de Chamadas, que contém informações detalhadas sobre as chamadas originadas, terminadas ou que passam pela central (JOHANSSON, 2007). O CDR contém algumas informações importantes como o tempo médio da chamada, o tempo de inicio e fim da chamada, o ramal que originou a chamada, para qual ramal foi encaminhada a chamada, a hora exata em que foi atendida a chamada, o tempo tarifado, entre outras informações. Todas essas informações são muito importantes e especialmente voltadas para a tarifação. Essa informação, nos dias de hoje, em *softswitch* e PABX, são geradas nos formatos de CSV (*Comma-Separated Values*) ou salvos diretamente em banco de dados.

Em redes telefônicas convencionais existem duas entidades diferentes, uma que gera e outra que processa os CDR's. Eles são inicialmente gerados pela rede telefônica e posteriormente coletados por um agente externo que valida, formata e depois envia para processamento. Esse processo também é conhecido como mediação. No fim esse é um processo trabalhoso dependendo o volume de chamadas que existe na central telefônica. Esses dados gerados são a renda de uma operadora telefônica e para evitar perda de dados geralmente são mediados em *hardware* tolerantes a falha e em *hardware* em paralelo.

2.1.2 Sistemas existentes

Existem alguns sistemas que já fazem a tarifação para o FreeSWITCH. Alguns são de código aberto e outros fechados. Os principais de código aberto são o ASTPP e o vBilling.

O ASTPP, dentre os pesquisados, aparenta ser o mais completo, pois ele tem várias funções que funcionam no FreeSWITCH, no Asterisk e OpenSIPS. Ele faz toda a parte de controle da conta dos clientes, sendo que esses podem ter contas pré e pós-pagas. As faturas para os clientes podem ser geradas em vários ciclos diferentes sendo diariamente ou até anualmente. Também há um módulo de revenda onde a revenda pode ter suas próprias tarifas, não ficando limitada ao preço imposto de quem ela está revendendo. O ASTPP também faz integração com outros sistemas de faturamento e exporta os dados de faturamento para CSV, que pode ser utilizado por outros sistemas também. Faz cálculos de rota de menor custo e *failover*[2]. Tem controle de DID (*Direct Inward Dialing*), que em português é denominado DDR (Discagem Direta a Ramal),

[2]quando um tronco cai ou está saturado e manda para um outro tronco livre

para os clientes.

O vBilling tem várias funções parecidas com as do ASTPP, como o modo de tarifação pré e pós-pago. Porém ele é mais focado na parte de controle do sistema, criando administradores de múltiplos níveis, assim como as revendas. Ele mostra também maior controle nos clientes, tendo controle em cima de *codecs* que vão ser utilizados e autenticação por IP. Também conta com algumas estatísticas, como as de *gateways* e CDR.

2.2 FreeSWITCH

FreeSWITCH foi criado por Anthony Minessale, que também desenvolveu aplicações para o Asterisk. O FreeSWITCH é definido como um *softswitch*, a definição de *softswitch* é de uma aplicação que faz uma ponte entre chamadas de telefonias publicas e chamadas VoIP separando a parte de controle de chamadas do *gateway* de mídia (ROUSE,). Ele teve a ideia de criar um *softswitch* que fosse modular, mesmo na época já sendo conhecido por desenvolver aplicações para Asterisk e sabendo que o mesmo já estava bem difundido entre uma das melhores soluções de código aberto para VoIP. Ele foi anunciado em 2006, o que causou bastante repercussão, pois o Asterisk já estava bastante estável e estava no mercado desde 1999. E em 2008 o FreeSWITCH teve sua versão 1.0 lançada.

Anthony Minessale começou a desenvolver o FreeSWITCH, após desenvolver muito para o Asterisk. Nesse tempo de desenvolvimento ele relata que mesmo o Asterisk tendo uma estrutura modular, ele tem dependências em cima de alguns módulos, o que significa que o mesmo não inicia sem alguns módulos específicos. Ele cita como exemplo o Asterisk 1.2 que não é possível iniciar o mesmo sem os módulos `app_dial` e `res_features` pois códigos utilizados no núcleo da aplicação estão dentro desses módulos (MINESSALE, 2008). Então ele propôs reescrever o Asterisk, o que iria dar um grande trabalho e foram poucas pessoas que aceitaram a ideia. Então no terceiro trimestre de 2005 ele resolveu fazer por si só o que ele queria que fosse o Asterisk 2.0 e começou o FreeSWITCH.

Só o que foi citado anteriormente, já mostra que há diferenças nos projetos. O FreeSWITCH foi concebido para ser totalmente modular e não ter problemas como o citado anteriormente do Asterisk 1.2, onde ele é modular mas não funciona sem certo módulos. Também acontece que o Asterisk tem maioria do seu código aberto e podem ser facilmente mal utilizado ou transpassados. Já o FreeSWITCH tem um núcleo fechado, com ações que são chamadas por seus módulos.

2.2.1 Arquitetura

O FreeSWITCH foi concebido para ser totalmente modular. Isso quer dizer que ele tem módulos que seriam as pontas do sistema. Toda comunicação com o FreeSWITCH é feita pelos módulos. Por exemplo, na Figura 2.1 tem as bolas menores, que são os módulos. Cada um desses módulos se refere a uma implementação diferente de um grupo maior. Na Figura 2.1 tem as bolas referentes ao G711, G729 e G722 que são implementações de módulos de *Codec* de áudio. Tem XML e ENUM, que são implementações de *Dialplans* (Planos de Discagem). Então basicamente existem grupos de módulos e implementações diferentes para cada um desses grupos, podendo atender de formas diferentes para fazer a mesma função.

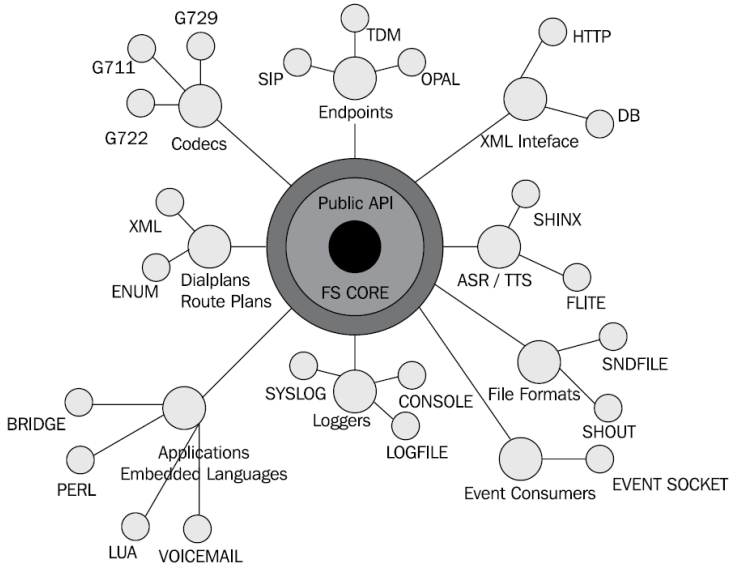

Figura 2.1: Diagrama de módulos do FreeSWITCH (MINESSALE, 2013)

Existem vários módulos no FreeSWITCH, como mostra a Figura 2.1, cada módulo tem sua função dentro do sistema como um todo. O sistema pode ser inicializado só com os módulos essenciais para o funcionamento (*Endpoint*, *Dialplan* e *Codec*) ou pode inicializar junto com outros módulos escolhidos pelo administrador do sistema. Alguns dos módulos existentes são os seguintes:

- *Endpoint* - módulos de protocolos telefônicos como SIP, IAX, H.323, etc.
- *Dialplan* - decide para onde as chamadas vão ser encaminhadas, dependendo do número discado.
- *Codec* - faz conversão entre os vários formatos de áudio que podem ser utilizados
- *Application* - faz algumas ações do sistema, como tocar os áudios de URA.

- *Application Programming Inteface* (API) - funções que recebem algum dado de entrada e retorna outro dado, pode ser usado tanto pelos módulos como por aplicações externas.
- *File* - faz o trabalho de tocar os vários formatos de áudio.
- *Text-to-Speech* (TTS) - interface com módulos de TTS
- *Automated Speech Recognition* - interface com módulos de fala
- *Diretctory* - conecta serviços de diretório, como LDAP, a API
- *Chat* - conexão e conversão entre os protocolos de chat.
- *Say* - junta vários áudios diferentes os toca em sequencia, formando frases do sistema ou frases que o usuário define.
- *XML Interfaces* - utiliza XML para os CDR's (Registro Detalhado de Chamadas) entre outras aplicações.

Para fazer a ligação entre esses módulos, existe um núcleo (*core*) que faz a conversação entre todos os módulos, como mostra na figura 2.1. Com isso os módulos não se conversam entre si diretamente. Então caso um módulo como o de *Endpoint* necessite de um codificador em específico, essa comunicação é feita diretamente com o núcleo do FreeSWITCH que verá se existe algum módulo de *Codec* que possa atender a necessidade do módulo de *Endpoint*. Caso existir, o módulo de *Codec* envia a resposta para o núcleo, que por sua vez envia para o módulo de *Endpoint*.

A vantagem do FreeSWITCH ser concebido de forma modular, é que a qualquer momento pode ser adicionado um módulo novo para adicionar uma nova funcionalidade ao sistema. Por exemplo, hoje um FreeSWITCH opera apenas com um *codec* especifico, porém surge a demanda de operar com um outro *codec*. Então só é necessário adicionar ou desenvolver um módulo de *Codec* para atender essa nova demanda.

2.2.2 Funcionamento

Como descrito na Seção 2.2.1, os módulos do FreeSWITCH não conversam entre si, todos conversam diretamente com o núcleo do FreeSWITCH, que vai procurar alguma forma de executar o que foi pedido a ele, se for possível. Vale ressaltar que pode existir mais de um tipo de cada módulo. Por exemplo, o FreeSWITCH pode ter 2 módulos de *Endpoint*, um para atender as necessidades com protocolo SIP (*Session Initiation Protocol*) e outro para as necessidades

do protocolo IAX[3] (*Internet Asterisk eXchange*). E como o módulo de *Codec*, que pode ter implementações para os *codecs* utilizados pelo mercado, como G711, G729, G722.

Uma forma bem simples de exemplificar o funcionamento do FreeSWITCH é mostrando como ocorre uma chamada SIP dentro de mesmo. Os módulos que vão ser envolvidos no exemplo são o de *Endpoint* e *Dialplan*. Como exemplo de módulo *Endpoint*, temos o `mod_sofia`, que utiliza o Sofia-SIP, um projeto de uma biblioteca em código aberto patrocinado pela Nokia. O FreeSWITCH faz uso dessa biblioteca para implementar o suporte a chamadas usando SIP. E como exemplo para o módulo de *Dialplan*, o que é mais utilizado é o XML *Dialplan*, que nada mais é do que instruções de plano de discagem dentro de arquivos XML para serem interpretadas pelo módulo.

Para iniciar uma chamada SIP é necessário que tenha no minimo 2 usuários registrados no FreeSWITCH. Esses usuários se registrarão através do módulo de *Endpoint*, que é o `mod_sofia`. Esse módulo é responsável entre toda sinalização SIP, tanto de entrada quanto de saída, ou seja, ele é a borda do FreeSWITCH para a sinalização SIP. Então tomando como exemplo 2 usuários registrados na extensão 1000 e 1001 e o usuário 1000 está ligando para o 1001. O terminal utilizado pelo usuário envia a ligação SIP para o FreeSWITCH e o `mod_sofia` recebe essa mensagem coleta os dados relevantes e cria uma chamada no núcleo do FreeSWITCH. O núcleo então diz que essa chamada está no estado de `ROUTING` (roteamento). Em seguida o núcleo chama o módulo de *Dialplan* para que procure por alguma instrução que se encaixe com o número discado, que foi 1001.

O módulo de Dialplan, que é o XML Dialplan, tem várias regras no arquivo XML. Nesse caso o XML deve ter uma instrução que tenha um `destination_number` que se encaixe com 1001, como no caso abaixo:

```
1 <extension name="exemplo">
2   <condition field="destination_number" expression="^1001$">
3     <action application="bridge" data="user/1001"/>
4   </condition>
5 </extension>
```

Na regra acima, existe a chave `<condition>`, nessa chave tem um parâmetro chamada `expression`. O número discado pelo usuário tem que ser igual ao que está ali ou atender as regras da expressão regular que pode ser passada, mais a frente tem um exemplo com expressão regular onde atende uma maior quantidade de números e não apenas um em especifico. Então dentro de `expression` existe no 1001, isso quer dizer que essa regra vai ser utilizada apenas

[3]Protocolo desenvolvido para estabelecer comunicação entre terminais com Asterisk.

quando um assinante discar para o número 1001. Dentro da chave `condition` tem a chave `action`, que diz qual ação que o FreeSWITCH deve tomar caso a condição seja aceita. Nesse caso a ação diz para fazer uma conexão entre o assinante que está fazendo a ligação, que será chamado de assinante A, com o assinante 1001, que será chamado de assinante B. O assinante A quando inicia a chamada, ele abre uma sessão com o FreeSWITCH. Depois que o assinante B atende essa chamada, o FreeSWITCH cria uma sessão com o assinante B. Nesse momento existem 2 sessões no núcleo, que é uma de entrada (sessão do assinante A) e uma de saída (sessão do assinante B). Com essas sessões, é o momento do FreeSWITCH executar a função que está na `action` do *Dialplan* e conectar as duas sessões para o assinante A e assinante B terem um fluxo de áudio entre si. Com isso, o papel desse *Dialplan* está terminado e a chamada entre os dois assinantes está concluída.

Caso o assinante B quisesse ligar para o assinante A, o fluxo dentro do FreeSWITCH seria o mesmo, trocando mensagens entre o `mod_sofia`, XML Diaplan e núcleo do FreeSWITCH. Porém no *XML Dialplan* seria necessário haver uma regra dizendo qual ação deveria ser tomada para esse caso ligassem para o assinante A. Ou poderia ser explorado o uso de expressões regulares, como dito anteriormente, utilizadas pelo XML Diaplan e criar uma regra mais genérica para qualquer número iniciado por 1 e que contenha mais 3 dígitos, como no exemplo abaixo:

```
1 <extension name="exemplo">
2   <condition field="destination_number" expression="^1(\d{3})$">
3     <action application="bridge" data="user/${destination_number}"/>
4   </condition>
5 </extension>
```

2.2.3 mod nibblebill

O `mod_nibblebill` é um módulo do FreeSWITCH de crédito e débito. Ele foi inicialmente desenvolvido por Darren Schreiber para preencher a necessidade de sistemas maiores de detectar fraudes em tempo real. A proposta é de realizar débito e crédito em tempo real em uma base de dados enquanto as chamadas estão em progresso (SCHREIBER, 2009). Os objetivos do projeto são os seguintes:

- Debitar crédito ou dinheiro das contas em tempo real.
- Permitir a tarifação com taxas diferentes durante uma única chamada.
- Permitir que o usuário saiba quando seu saldo está baixo, através de áudio por exemplo.

- Permitir que a chamada seja finalizada ou que seja encaminhada quando o saldo está esgotado.
- Permitir funções de tarifação para operar com várias chamadas concorrentes.

Não existe exemplos mostrando como é o real funcionamento desse módulo, porém em testes foi visto que o funcionamento dele é um pouco diferenciado de um tarifador mais usual. Um tarifador usual faria a tarifação cobrando uma taxa de conexão e depois faria a cobrança sobre a cadência. Essa taxa de conexão não existe no `mod_nibblebill`. Quando uma chamada é atendida o `mod_nibblebill` já faz a tarifação e assim que acaba o tempo da cadência global ele faz a tarifação novamente. O seu maior problema é que quando a chamada é desligada, ele faz uma tarifação novamente. Por exemplo, o `mod_nibblebill` está configurado para fazer a cobrança a cada 6 segundos, o assinante fala com outro durante 8 segundos, com isso o `mod_nibblebill` deveria cobrar o equivalente a 12 segundos, 6 segundos de quando a chamada foi atendida, mais 6 pois já passaram os 6 primeiros, porém ele cobra 18 segundos.

O `mod_nibblebill` faz parte dos módulos padrões do FreeSWITCH, porém ele não vem habilitado por padrão. Para habilitar ele no FreeSWITCH é necessário alguns passos, que estão documentados na wiki do FreeSWITCH[4]. Após instalado e inicializando junto com o FreeSWITCH, algumas configurações são necessárias no arquivo `nibblebill.conf.xml`, que é o arquivo que conta com as configurações do `mod_nibblebill`.

Dentro do arquivo é necessário configurar duas colunas em uma tabela do banco de dados, a primeira coluna dessa tabela que indica o crédito e a segunda indica a conta do usuário. Isso é necessário, pois o `mod_nibblebill` tem umas instruções próprias para serem executadas nos bancos de dados e necessita dessas informações para executar corretamente. No arquivo essas informações estão descritas da seguinte maneira:

```
1  <param name="db_table" value="USER"/>
2  <param name="db_column_cash" value="CREDIT"/>
3  <param name="db_column_account" value="ACCOUNT"/>
```

O parâmetro `db_table` é referente a tabela que será executada a instrução, `db_column_cash` se refere a coluna onde serão salvos os créditos e `db_column_account` se refere a coluna que tem a informação de qual a conta do usuário. Também é possível configurar dentro desse mesmo arquivo, algumas outras ações e informações, como a cadência global[5], quando que deve avisar que os créditos do usuário estão acabando, qual ação tomar quando estiver acabando o crédito,

[4] https://wiki.freeswitch.org/wiki/Mod_nibblebill

[5] De quanto em quanto tempo será executada a instrução de débito/crédito

qual o valor deve ser considerado para finalizar chamadas e também existe como configurar um valor máximo por chamada.

Com toda parte do `mod_nibblebill` configurado, é necessário fazer ajustes no plano de discagem, no módulo de *Dialplan*, ou nos usuários, no módulo de *Endpoint*. Existem alguns parâmetros que podem ser passados tanto no módulo de *Dialplan*, quanto no módulo de *Endpoint* e esses parâmetros que vão passar valores para o `mod_nibblebill` fazer os débitos na conta do usuário. O `nibble_rate` é o parâmetro que vai definir qual o valor que vai ser cobrado cada cadência e o `nibble_account` é a conta ou um identificador do usuário que vai estar no banco de dados que será debitado.

Para configurar tanto os valores das chamadas, quando a conta do usuário dentro do módulo de *Endpoint*, ficaria assim o arquivo XML do usuário, por exemplo:

```
<user id="1000">
  <params>
    <param name="password" value="1000"/>
  </params>
  <variables>
    <variable name="nibble_rate" value="0.15"/>
    <variable name="nibble_account" value="1000"/>
    <variable name="user_context" value="default"/>
  </variables>
</user>
```

O XML acima diz que toda a conta ou identificador do usuário 1000 é o número 1000 e que o valor a ser cobrado desse usuário será de 0.15 a cada vez que atingir o tempo que é configurado na cadência global. Porém configurado dessa maneira, todas as chamadas desse usuário serão tarifadas com o valor de 0.15. Para caso de configurações de chamadas com valores diferentes como praticado pelas operadores, que tem valores diferentes para DDD e DDI por exemplo, é necessário configurar o parametro `nibble_rate` dentro do módulo de Dialplan. Pegando o exemplo usado no capitulo 2.2.2 e mudando para chamar esse módulo de tarifação:

```
<extension name="exemplo">
  <condition field="destination_number" expression="^1d{3}$">
    <action application="set" data="nibble_rate=0.10"/>
    <action application="bridge" data="user/${destination_number}"/>
  </condition>
</extension>
```

Nesse XML, só foi adicionado o `nibble_rate`, pois só era necessário adicionar o valor. No caso, foi apenas adicionar um ação nova dentro do plano de discagem, antes de ele realmente fazer o `bridge`, que essa ação vai adicionar essa configuração, sobrescrevendo a anterior que definia o `nibble_rate` com o valor de 0.15. O `nibble_account` não é necessário configurar, pois ele já vem configurado dentro do próprio usuário, mas caso queira configurar seria com a mesma ação, trocando apenas os dados que estão dentro do `data` para `nibble_account=2000` por exemplo.

2.2.4 Mod XML Curl

Este módulo é utilizado para obter configurações do do FreeSWITCH via HTTP. Este módulo é chamado pelo FreeSWITCH sempre que ele precisar de alguma configuração que normalmente é lida de um XML (BOTELER, 2014). É recomendado a utilização desse módulo em casos de reais necessidades, pois com ele será necessário se preocupar com segurança de dados e redes por exemplo. Pois os dados vão ser obtidos através da rede, o que faz que esses dados, como usuário e senha de um assinante, possa ser interceptado e utilizado por alguma pessoa mal intencionada. Essa preocupação com segurança deve ser tomada, pois o FreeSWITCH fará requisições para uma aplicação, essa aplicação deve interpretar os pedidos e retornar os dados do pedido. Abaixo ficam alguns casos de uso comum dessa aplicação:

- Ter várias instâncias do FreeSWITCH sem ter que manter várias configurações.
- Gerenciamento centralizado das configurações.
- A configuração fica dinâmica, pois a aplicação pode buscar dados de um banco de dados e montar um XML diferente para cada situação.
- Possibilita a mudança de configuração de alguns módulos, como o de Dialplan e Endpoint.

O módulo tem o seu funcionamento através de mensagens trocadas via HTTP entre o `mod_xml_curl` e um servidor de aplicação. Para melhor exemplificar, supondo que tenha um assinante A fazendo uma ligação para um assinante B. Sem entrar em detalhes de todo o percurso da chamada, em um momento dessa ligação o FreeSWITCH necessita saber o plano de discagem para o número do assinante B, que foi discado pelo assinante A. Nesse momento, supondo que o `mod_xml_curl` esteja configurado no FreeSWITCH, o núcleo do FreeSWITCH pede para o `mod_xml_curl` esse plano de discagem. O `mod_xml_curl`, por sua vez, envia uma

mensagem via HTTP para uma aplicação pedindo esse plano de discagem. Nessa mensagem enviada contém vários dados da chamada, do assinante A e do assinante B. Em seguida a aplicação responde para o `mod_xml_curl` com um XML, esse XML pode ser tanto o XML com a resposta esperada em caso de sucesso na busca da informação, quanto um XML padrão de falha em caso de falha na busca da informação.

A princípio ele é um módulo que não inicia junto com o FreeSWITCH, porém ele já vem com ele. Depois de instalado, é necessário configurar o módulo e mostrar onde que ele deve enviar uma mensagem para esperar um XML de resposta. O módulo se divide em 4 seções dispostas da seguinte maneira:

- *configuration* - XML para configuração de módulos, como por exemplo o `mod_sofia`.
- *directory* - Diretório de usuário utilizados para autenticação.
- *dialplan* - XML destinado para o módulo XML Dialplan, que configura o plano de discagem.
- *languages* - Frases para módulos de fala.

O `mod_xml_curl` tem um arquivo chamado `xml_curl.conf.xml`. Dentro deste arquivo é necessário apenas informar qual seção vai se utilizar do `mod_xml_curl` e passar qual o endereço que ele vai enviar essas mensagens HTTP. No exemplo abaixo está exemplificando para a seção de *directory* e de *dialplan*:

```
1  <configuration name="xml_curl.conf" description="cURL XML Gateway">
2    <bindings>
3
4      <binding name="directory">
5        <param name="gateway-url" value="http://exemplo:80/freeswitch/directory"
             bindings="directory"/>
6      </binding>
7
8      <binding name="dialplan">
9        <param name="gateway-url" value="http://exemplo:80/freeswitch/dialplan"
             bindings="dialplan"/>
10     </binding>
11
12   </bindings>
13 </configuration>
```

Com isso, o FreeSWITCH sempre que precisar de qualquer informação referente aos usuários que existem no sistema, ou quando vai buscar alguma instrução do plano de discagem, vai enviar uma mensagem de requisição para os endereços citados no exemplo e a aplicação desse endereço enviar um XML de sucesso ou falha, dependendo do resultado do processamento dessa mensagem.

3 *Sistema de Tarifação para FreeSWITCH*

Este trabalho implementa um sistema de tarifação para o FreeSWITCH. O sistema conta com cadastro de usuários, administradores, plano de discagem, tarifa de compra, tarifa de venda, adicionar créditos para os usuários, coleta de CDR e tarifação pré e pós paga.

Todo o sistema foi feito com ferramentas de código aberto e a integração de módulos já existentes dentro do próprio FreeSWITCH. Ele tem dados cadastrados dentro de um banco de dados, que são cadastrados através de uma inteface gráfica, e os módulos do FreeSWITCH fazem tarifação e interpretação desses dados vindos do banco de dados. A tarifação pré paga é feita em tempo real, pois caso os créditos do usuário estejam acabando, é necessário cortar a chamada. Já a tarifação pós paga é feita manualmente quando o usuário quiser, através de uma interface do sistema.

As ferramentas necessárias para o desenvolvimento do projeto são:

- Ubuntu Linux 32 bits - Sistema operacional escolhido e que suporta todas as ferramentas necessárias;
- FreeSWITCH - Softswitch escolhido para desenvolver a ferramenta;
- mod_nibblebill - Módulo do FreeSWITCH que faz a tarifação pré paga;
- mod_xml_curl - Módulo do FreeSWITCH para que o plano de discagem e usuários pudessem ser carregados dinamicamente;
- Java Enterprise Edition 6 - Plataforma de programação utilizada para criar as interfaces e regras de negócio;
- Eclipse Kepler Service Release 2 - IDE (*Integrated Development Environment* ou Ambiente de Desenvolvimento Integrado) Java EE para desenvolvimento Web, ferramenta de código aberto e mantida pela *The Eclipse Foundation*;

3.1 Arquitetura do Sistema

O sistema foi concebido visando apenas uma interface para os usuários do tipo administrador, não existe uma interface, ou similar, para os usuários. A Figura 3.1 demonstra como foi pensada a arquitetura do sistema. A única forma de um usuário interagir com o sistema é através de um telefone IP ou *softphone*. A conexão do cliente é direta com o FreeSWITCH, eles podem estar conectados diretamente dentro da mesma rede, ou caso o FreeSWITCH esteja exposto na internet, eles podem se conectar externamente também.

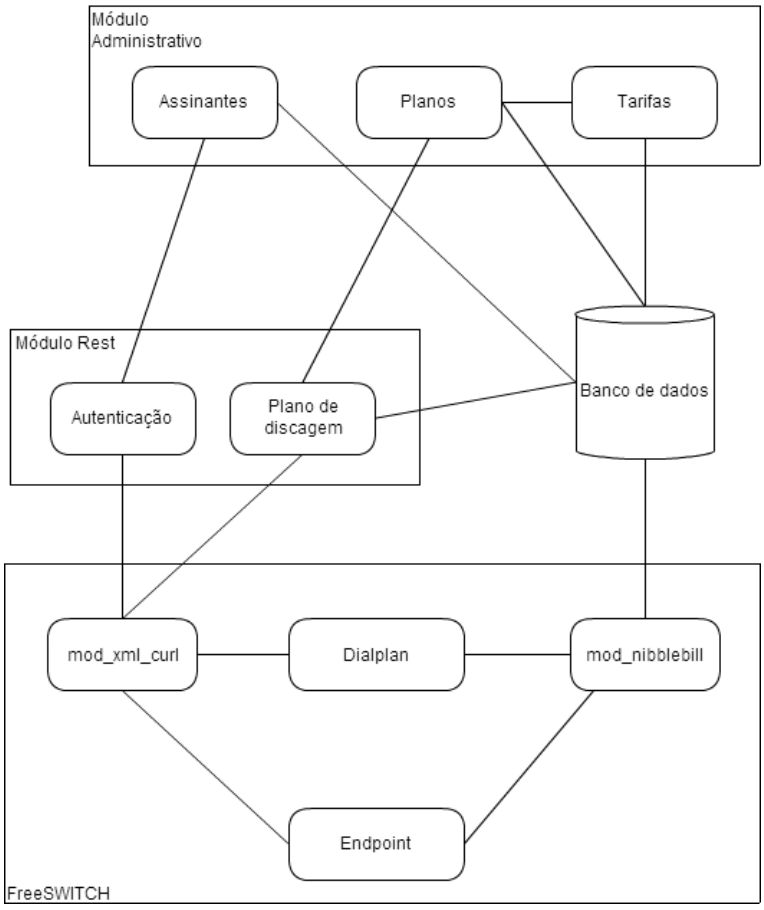

Figura 3.1: Diagrama de blocos mostrando a arquitetura do sistema

Na visão do administrador, o sistema é mais complexo, pois ele terá acesso a todas as partes do sistema. O administrador, a princípio terá a visão do módulo de administração, pois após a configuração inicial do sistema, não é necessário mexer no FreeSWITCH. Só será necessário configurar algo no FreeSWITCH, caso for adicionar novas rotas de saída.

O módulo administrativo será feito o cadastro de dados para alimentar o FreeSWITCH. Todas parte de usuário, tarifas, plano de discagem é cadastrada no próprio módulo administrativo. O FreeSWITCH então fica configurado através do `MOD_XML_CURL` para buscar informações dos usuários e dos planos de discagem através de um Web Service configurado no módulo adminis-

trativo.

Para exemplificar o funcionamento do sistema, existe um assinante A querendo se autenticar no sistema. O FreeSWITCH recebe essa requisição e repassa ao módulo administrativo para retornar o XML desse usuário. O módulo administrativo por sua vez, interpreta o *REQUEST* que o FreeSWITCH manda e utiliza somente as informações que vem nos parâmetros `user`, `domain` e `section`. O `user` identifica quem é o usuário que fez a requisição, `domain` o domínio do usuário que vai ser adicionado no XML e `section` é a seção, descrita na seção 2.2.4. Caso o usuário exista na base de dados, ele retorna o seguinte XML:

```
<document type="freeswitch/xml">
  <section name="directory">
    <domain name="domain.exemplo.com">
      <params>
        <param name="dial-string" value="{presence_id=${dialed_user}@${
            dialed_domain}}${sofia_contact(${dialed_user}@${dialed_domain})}"/>
      </params>
      <groups>
        <group name="default">
         <users>
          <user id="1000">
            <params>
              <param name="password" value="1000"/>
            </params>
          </user>
         </users>
        </group>
      </groups>
    </domain>
  </section>
</document>
```

As *tags* `section` e `domain` são referentes ao que foi explicado anteriormente. A primeira *tag* `param` são os parâmetros desse usuário, no exemplo tem apenas o dial-string, pois é o único obrigatório para execução de uma chamada. Depois a *tag* importante é a `user`, que retorna o usuário que foi passado e como parâmetro desse usuário aparece sua senha. Esse XML vai passar parâmetros para o FreeSWITCH manter esse usuário registrado.

No caso do plano de discagem, acontece o mesmo processo. O usuário inicia uma chamada, o FreeSWITCH chama o *Web Service* a procura do XML para saber as instruções que deve executar. A diferença é que agora o XML de retorno não vai ter a *tag* `section` do tipo `directory`

e sim `dialplan`. Também no `RESQUEST` virão outras informações, agora referente a discagem do usuário. O XML de retorno deve ser algo parecido com o XML apresentado a seguir.

```
<document type="freeswitch/xml">
  <section name="dialplan" description="Exemplo Plano de Discagem">
    <context name="default">
      <extension name="1001">
        <condition field="destination_number" expression="^1000$">
          <action application="bridge" data="user/1001"/>
        </condition>
      </extension>
    </context>
  </section>
</document>
```

As tags a partir do `extension` não diferem em nada do que foi explicado no seção 2.2.2 em que é falado sobre o funcionamento do FreeSWITCH. A única diferença é que o sistema vai retornar alguns dados de tarifação, que é explicado no seção 3.3, caso o sistema possua tarifação pré-paga.

Porém ainda existe o caso de não existir o dado que o FreeSWITCH procura. Nesse caso ele retorna um XML padrão para indicar que não foi achado, como o do exemplo a seguir.

```
<document type="freeswitch/xml">
  <section name="result">
    <result status="not found" />
  </section>
</document>
```

3.1.1 Casos de uso

Os casos de uso especificam os serviços oferecidos pelo sistema, ou seja, descreve como o serviço pode ser utilizado. Este documento auxiliam na hora do desenvolvimento. Os casos de uso devem ser feitos antes de realmente começar a desenvolver o projeto, pois definem como determinado serviço deve funcionar independente da linguagem e tecnologias escolhidas para o desenvolvimento do sistema.

Para fazer o modelo dos casos de uso é interessante primeiro estar bem definido como o que o sistema vai fazer e quem vai interagir com o sistema. Com isso é possível identificar os atores do sistema, que são as pessoas que vão interagir com o sistema e os casos de uso propriamente

ditos, que são as ações que o ator vai poder fazer. Não é necessário se preocupar em tentar mapear todos os casos inicialmente, pois é comum durante o desenvolvimento de um sistema existirem novas ideias. Porém quando for aparecendo novas ideias é interessante já documentar, pois ficara mais fácil a implementação no futuro.

Nas próximas seções ficarão os casos de uso identificados no sistema desenvolvido nesse projeto, eles seguem um modelo descrito no livro "Princípios de Análise e Projeto de Sistemas com UML", escrito por Eduardo Bezerra. Os atores identificados no projeto foram Administrador e Assinante.

Tarifação

Aviso de crédito baixo

Assinante	**Sistema**
Inicia uma chamada para assinante B.	
	Encaminha a chamada para assinante B caso o assinante tenha crédito.
Conversa com assinante até seus créditos estarem próximos do fim.	
	Detecta que o crédito está próximo do fim e manda sinal sonoro.

Encerrar chamada ao acabar crédito

Assinante	**Sistema**
Inicia uma chamada para assinante B.	
	Encaminha a chamada para assinante B caso o assinante que origina a chamada tenha crédito.
Conversa com assinante até seus créditos acabarem.	
	Detecta que o crédito chegou ao fim e finaliza a chamada.

Realizar tarifação pré paga

Assinante	**Sistema**
Inicia uma chamada para assinante B.	Verifica se o assinante que inicia a chamada tem crédito. Caso tenha, debita e inicia a chamada.
Conversa com assinante B durante determinado tempo.	A cada X segundos do tempo, debita o crédito para falar mais tempo.

Tarifas

Inserir Tarifa de Compra

Administrador	**Sistema**
Acessa a tela para inserir tarifa de compra.	Apresenta os campos Cadência, Valor e Descrição.
Preenche todos os campos apresentados.	Salva os campos no banco de dados.

Inserir Tarifa de Venda

Administrador	**Sistema**
Acessa a tela para inserir tarifa venda.	Apresenta os campos Cadência, Valor, Tarifa de compra e Descrição.
Preenche todos os campos apresentados.	Salva os campos no banco de dados.

Editar Tarifa de Compra

Administrador	**Sistema**
Acessa a tela com a lista das tarifas de compra.	
	Mostra lista com todas as tarifas de compra registradas no sistema.
Seleciona a tarifa que quer editar.	
	Apresenta os campos Cadência, Valor e Descrição com os valores preenchidos.
Edita os valores que achar necessário.	
	Salva os campos no banco de dados.

Editar Tarifa de Venda

Administrador	**Sistema**
Acessa a tela com a lista das tarifas de venda.	
	Mostra lista com todas as tarifas de venda registradas no sistema.
Seleciona a tarifa que quer editar.	
	Apresenta os campos Cadência, Valor Tarifa de compra e Descrição com os valores preenchidos.
Edita os valores que achar necessário.	
	Salva os campos no banco de dados.

Remover Tarifa de Compra

Administrador	**Sistema**
Acessa a tela com a lista das tarifas de compra.	
	Mostra lista com todas as tarifas de compra registradas no sistema.
Seleciona a tarifa que quer remover.	
	Apresenta janela para confirmar a remoção.
Exclui a tarifa do banco de dados.	

Remover Tarifa de Venda

Administrador	**Sistema**
Acessa a tela com a lista das tarifas de venda.	
	Mostra lista com todas as tarifas de venda registradas no sistema.
Seleciona a tarifa que quer remover.	
	Apresenta janela para confirmar a remoção.
Exclui a tarifa do banco de dados.	

Visualizar Tarifas

Administrador	**Sistema**
Acessa a tela com a lista das tarifas de venda.	
	Mostra lista com todas as tarifas de venda registradas no sistema.

Planos

Inserir Planos

Administrador	**Sistema**
Acessa a tela para inserir os planos.	
	Apresenta o campo Nome do plano e o tipo do plano (pré ou pós-pago).
Preenche os campos apresentados e seleciona o tipo do plano.	
	Salva os dados no banco de dados.

Editar Planos

Administrador	**Sistema**
Acessa a tela com a lista de planos.	
	Apresenta lista com todos os planos registrados no sistema.
Seleciona o plano que deseja editar.	
	Apresenta a tela de cadastro com os dados do plano selecionado preenchidos em seus campos.
Edita os dados.	
	Salva os dados no banco de dados.

Remover Planos

Administrador	**Sistema**
Acessa a tela com a lista de planos.	
	Apresenta lista com todos os planos registrados no sistema.
Seleciona o plano que deseja remover.	
	Remove o plano do sistema.

Visualizar Planos

Administrador	**Sistema**
Acessa a tela com a lista de planos.	
	Apresenta lista com todos os planos registrados no sistema.

Créditos

Inserir Crédito para Assinante

Administrador	**Sistema**
Acessa tela para inserir créditos.	
	Apresenta os campos de Assinante e Valor.
Preenche campo de Assinante e campo Valor com um valor positivo.	
	Salva dados no banco de dados e adiciona créditos ao assinante selecionado.

Remover crédito de Assinante

Administrador	**Sistema**
Acessa tela para inserir créditos.	
	Apresenta os campos de Assinante e Valor.
Preenche campo de Assinante e campo Valor com um valor negativo.	
	Salva dados no banco de dados e remove créditos ao assinante selecionado.

Assinante

Inserir Assinante

Administrador	**Sistema**
Acessa a tela para inserir assinante.	
	Apresenta o campo Nome completo, E-mail, Usuário do FreeSWITCH, Senha do FreeSWITCH, CallerID do FreeSWITCH e o Plano.
Preenche os campos apresentados.	
	Salva os dados no banco de dados.

Editar Assinante

Administrador	**Sistema**
Acessa a tela com a lista de Assinantes.	
	Apresenta lista com todos os Assinantes registrados no sistema.
Seleciona o Assinante que deseja editar.	
	Apresenta a tela de cadastro com os dados do Assinante selecionado preenchidos em seus campos.
Edita os dados.	
	Salva os dados no banco de dados.

Remover Assinante

Administrador	**Sistema**
Acessa a tela com a lista de Assinantes. Seleciona o Assinante que deseja remover.	Apresenta lista com todos os Assinantes registrados no sistema. Remove o Assinante do sistema.

Visualizar Assinante (Ator: Administrador)

Administrador	**Sistema**
Acessa a tela com a lista de Assinantes.	Apresenta lista com todos os Assinantes registrados no sistema.

Relatórios e Fechamentos

Gerar Relatório de chamadas por assinante

Administrador	**Sistema**
Acessa a tela para geração do relatório. Gera o relatório e apresenta *download*.	Apresenta tela com os campos de Assinante, Data Inicial e Data Final.

3.1.2 Modelo Estrutural

O sistema foi inteiramente modelado para trabalhar com um banco de dados relacional como o MySQL, PostgreSQL, SQL Server ou Oracle por exemplo. A estrutura foi feita pensando basicamente nos dados que seriam necessários para o funcionamento básico do sistema. Todas acabam tendo um relacionamento, pois o sistema foi dividido para facilitar o entendimento e também para evitar a repetição de dados. Por exemplo, os dados das tabelas de TARIFA_COMPRA e TARIFA_VENDA poderiam estar como dados dentro da tabela PLANO_DE_DISCAGEM, porém da forma que foi feito, não há a necessidade de repetir os dados a cada registro de um plano de discagem. Assim evita dados repetidos e facilita a mudança de dados.

A tabela FREESWITCH_CDR é a única tabela que não está diretamente ligada a nenhuma

tabela pois ela é um tabela alimentada pelo FreeSWITCH. A única interação do usuário com essa tabela é gerar relatórios. Apesar dela ter uma relação com a tabela USUARIO por conta do CallerID, foi decidido não transformar isso em chave primária e fazer a ligação entre as tabelas.

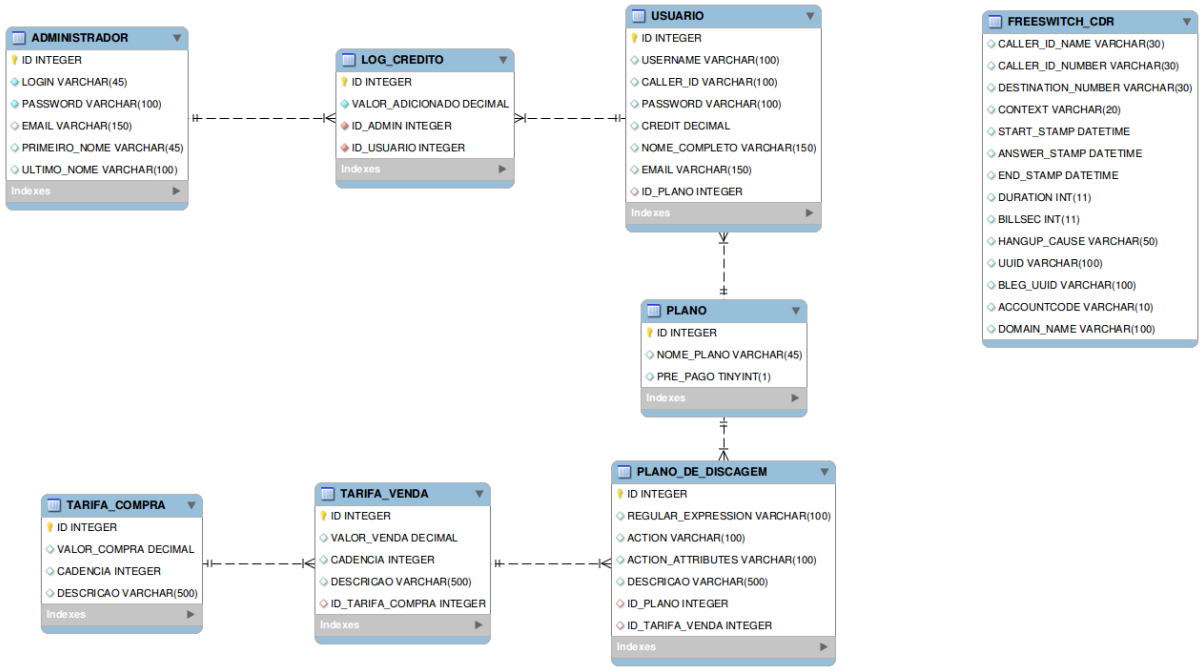

Figura 3.2: Modelagem do Banco de dados do sistema

O cadastro dos dados não é feito diretamente no banco de dados. Todo o cadastro de dados é feito através de um software com interfaces modeladas para esse fim e que respeitam as regras impostas pelo banco de dados.

3.2 Cadastro dos dados

O cadastro dos dados é feito de forma manual pelo administrador do sistema. Mas nada impede que os dados sejam importados diretamente no banco de dados, desde que respeite a estrutura do banco de dados. Para o cadastro ficar o mais completo possível, é necessário seguir uma sequência lógica, pois alguns dados são necessários para o cadastro de outros, como visto na Figura 3.2. A sequência de cadastro é a mesma disposta nos seções a seguir.

3.2.1 Administrador

O cadastro dos dados de Administrador é feita de uma forma simplificada, não tem dados muito detalhados sobre o administrador, que para uma segurança maior talvez fosse necessário, mas como o foco do projeto não era esse então foi feito dessa forma. Esse usuário vai acessar o

sistema para adicionar ou alterar qualquer dado. Para acessá-lo é no menu Sistema, que está no cabeçalho. Como pode ser visto na Figura 3.3.

Figura 3.3: Tela de cadastro de um usuário do tipo administrador

Todos os campos da Figura 3.3 que contém um "*"devem ser preenchidos. Os campos Primeiro Nome e Último Nome são dados para identificação do usuário, que é o nome que aparece no cabeçalho do sistema. O campo Login é o campo para cadastro do usuário que vai dar acesso ao sistema e Senha é sua respectiva senha. O campo e-mail ficou para implementações futuras, esse campo não está sendo utilizado ainda. Porém caso surgisse a demanda de enviar e-mails de notificação para os administradores, o campo já existe no sistema.

3.2.2 Planos

O cadastro dos planos é o mais simples do sistema. Ele serve apenas para criar uma maneira simples de agrupar os planos de discagem e facilitar o cadastro dos dados no usuário, eliminando assim a necessidade de fazer várias conexões iguais entre usuário e plano de discagem. Ele também indica se determinado usuário tem tarifação do tipo pré ou pós paga. Para acessar o cadastro dos planos é no menu FreeSWITCH, pois foi considerada uma parte de configuração do mesmo.

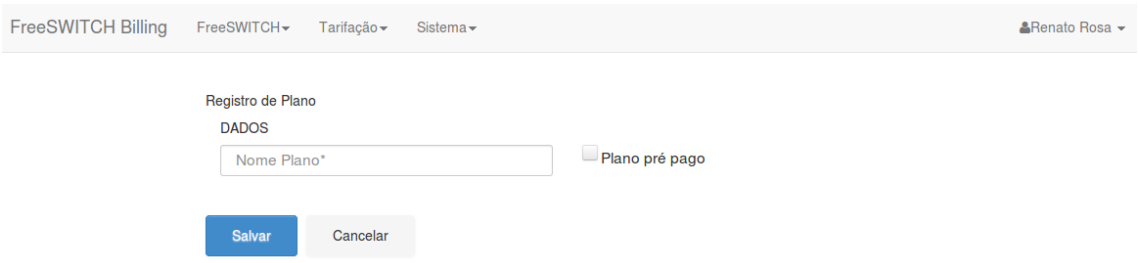

Figura 3.4: Tela de cadastro de um plano dentro do sistema

Então o único campo necessário para o cadastro é o nome para esse plano e marcar se esse plano é Pré-Pago, caso não for é só deixar o campo desmarcado.

3.2.3 Tarifa de compra

A tarifa de compra foi criada para caso futuramente queira ser feito algum relatório financeiro em cima do sistema. Nada mais é do que o cadastro de dados que vão dizer qual o valor que o administrador do sistema está pagando por uma rota de saída, por exemplo. Esse dado vai ser associado com a tarifa de venda, que por sua vez está associado ao plano de discagem. Ou seja, ele diz apenas qual o valor de custo do tempo daquela chamada, não o valor de venda final. O cadastro é acessado através do menu de Tarifação.

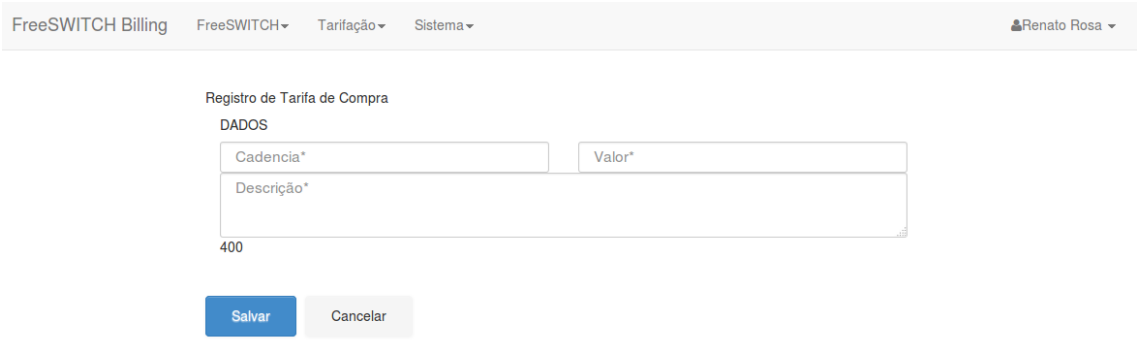

Figura 3.5: Tela de cadastro de uma tarifa de compra

As informações necessárias para o cadastro são a cadência, que é de quanto em quanto tempo é cobrado essa chamada (a cadência é melhor explicada no seção 3.3). Também tem o valor monetário dela e uma breve descrição para facilitar a identificação dessa tarifa de compra.

3.2.4 Tarifa de venda

A tarifa de venda é a tarifa que vai ser cobrada do usuário final. O cadastro e campos tem a disposição semelhante ao da Tarifa de Compra a única diferença é que existe um campo no meio que é necessário relacionar uma Tarifa de Compra a uma de venda. Isso é necessário, pois facilita a busca na geração de futuros relatórios e também evita de cadastrar os dados duas vezes dentro do plano de discagem.

Para o cadastro é necessário a cadência, que é medida em segundos. Valor, que é o valor cobrado do usuário final. Uma tarifa de compra, para saber quanto que está sendo pago por aquela chamada e uma breve descrição da tarifa.

3.2.5 Planos de discagem

O cadastro do plano de discagem é a parte mais complexa do cadastro de dados do sistema. Os dados aqui cadastrados requerem um pouco de conhecimento de expressões regulares, pois

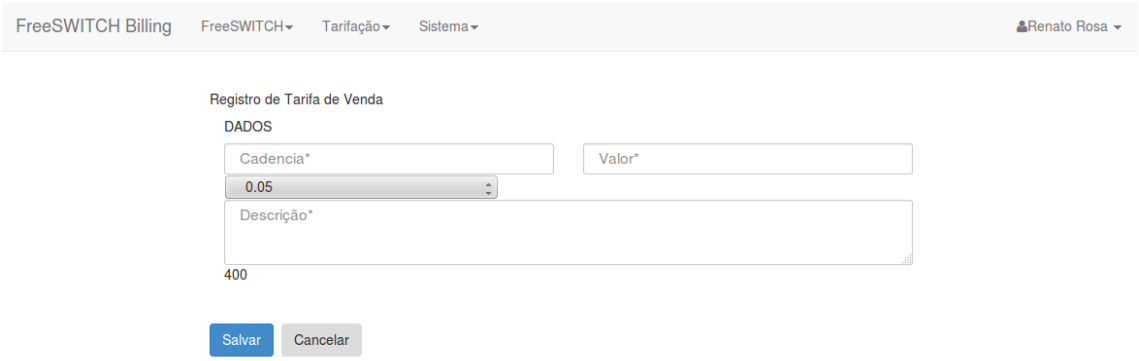

Figura 3.6: Tela de cadastro de uma tarifa de venda

precisa implementar de forma dinâmica um grupo de números, por exemplo, todos os números que começarem com 9 e tiverem mais 7 dígitos farão a mesma ação dentro do sistema. Assim não é necessário cadastrar uma regra para cada número dentro do sistema.

Também é necessário o conhecimento das ações que podem ser usadas no próprio FreeSWITCH. Como nos testes não foram usados outros módulos do FreeSWITCH, foi utilizado apenas a ação de *Bridge*. Mas nada impede de outras ações serem executadas, desde que obedeçam as regras do próprio FreeSWITCH. O conhecimento de como colocar os atributos da ação também é necessário, pois sem a configuração correta, algumas regras do plano de discagem podem não funcionar corretamente.

Figura 3.7: Tela de cadastro de um plano de discagem

O primeiro campo é a escolha de qual plano esse plano de discagem faz parte, esse plano deve estar previamente cadastrado no sistema. O segundo é a tarifa de venda e a cadência, sendo que a cadência está salva diretamente na tarifa de venda, como descrita na seção 3.2.4. Também é necessária uma ação, como já citada como exemplo o Bridge. O Atributos da Ação, são os atributos necessários para a realização daquela ação, podem ser utilizadas variáveis reconhecidas pelo FreeSWITCH que vão funcionar corretamente. Depois vem a Expressão Regular, que é a expressão dentro do padrão utilizado pelo FreeSWITCH. E uma breve descrição sobre o

plano de discagem.

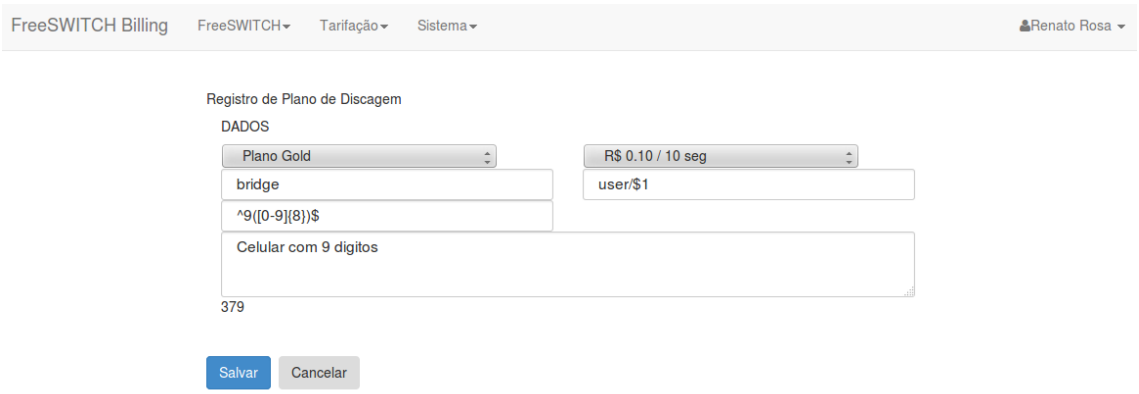

Figura 3.8: Tela de cadastro de um plano de discagem já preenchida

Na Figura 3.8 tem um plano de discagem de exemplo. Nesse exemplo foi desconsiderado DDD ou qualquer outro prefixo, que em uma regra mais complexa deve ser considerado. A regra consiste em identificar um número de celular de 9 dígitos e fazer um bridge para esse número. Então explicando a parte da expressão regular ^9([0-9]{8})$, o "^" marca o começo da expressão e diz que a expressão deve começar exatamente com o que vem a seguir. Então ^9 diz que o número que vai encaixar com essa regra, obrigatoriamente começa com 9, não tem como um número 8912341234, por exemplo, encaixar nessa regra. Depois entre parênteses tem a expressão [0-9]{8}, essa expressão diz que aceita qualquer número entre 0 e 9 e ela deve ser repetida 8 vezes. Assim é garantido que qualquer número de 9 dígitos, sendo que o primeiro seja um número 9, vá utilizar essa regra do plano de discagem.

Também tem nesse plano de discagem de exemplo na parte de atributos o seguinte dado `user/$1`. O user indica de onde ele deve buscar o usuário, que no caso é um usuário que está diretamente conectado no FreeSWITCH. O $1 é uma variável comum do FreeSWITCH, que vai ser substituída pelo número enviado para a regra. Então caso o usuário esteja ligando para o número 999999999, o FreeSWITCH internamente faria a regra ficar `user/999999999`.

3.2.6 Assinantes

O cadastro do assinante consiste no cadastro do usuário final do sistema. A princípio é um cadastro simples, sem muitas informações sobre o assinante, pois ele foi feito apenas para o funcionamento do FreeSWITCH. Porém nada impede de no futuro o sistema ser expandido para ter cadastro de outros dados, como e-mail, endereço e demais informações que forem necessárias para futuras funcionalidades.

Os únicos campos do cadastro que não são necessários para o FreeSWITCH são os campos

de Nome Completo e E-mail. O campo Nome Completo é o nome do assinante. Esse campo vai ser utilizado na geração do faturamento pós pago. Já o E-mail é apenas para ter um contato do assinante. O restante dos campos são todos necessários para o FreeSWITCH, eles que vão apontar os dados que o assinante vai utilizar dentro do sistema. O campo Usuário FreeSWITCH seria o *login* que o assinante vai utilizar no seu *softphone*, ou telefone IP, para se autenticar no FreeSWITCH. Isso junto com a senha que também é fornecida no campo Senha FreeSWITCH. Já o CallerID FreeSWITCH é uma chave única que o assinante vai receber dentro do próprio FreeSWITCH. No Plano, temos o plano que o assinante vai utilizar. O plano vai definir quais as rotas e valores que vão ser utilizados para esse assinante.

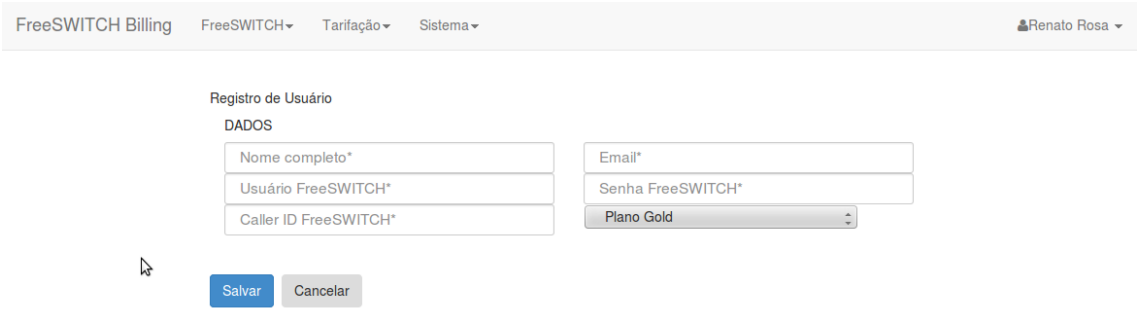

Figura 3.9: Tela de cadastro de um assinante

3.3 Tarifação Pré Paga

A tarifação pré paga é a mais complicada do sistema. Ela tem que ocorrer em tempo real, pois caso os créditos do usuário sejam insuficientes a chamada não deve continuar. O diagrama de fluxo da Figura 3.10 mostra exatamente como deve ocorrer a chamada, com tarifação, desde o seu começo até o fim dela.

Seguindo o fluxograma, é possível notar que só começa a ser descontado crédito do usuário A (usuário que origina a chamada) a partir do momento que o usuário B (usuário que recebe a chamada) atende a mesma. Antes disso há operações do próprio FreeSWITCH para determinar onde a chamada deve ser encaminhada, se há rota para isso e se o usuário A atende todas as questões que aparecem na Figura 3.10.

A tarifação é feita em cima do tempo de uso. No sistema tem a cadência, geralmente dada em segundos, cadastrada nas tarifas de compra e venda. Geralmente as operadoras utilizam a cadência com dois tempos, como 60/30, o que não é o caso do tarifador aqui apresentado. Isso quer dizer que existe uma “taxa de conexão” e a cadência como apresentada anteriormente. O 60 seria a taxa de conexão, então assim que o usuário B atende a ligação é cobrado do usuário A

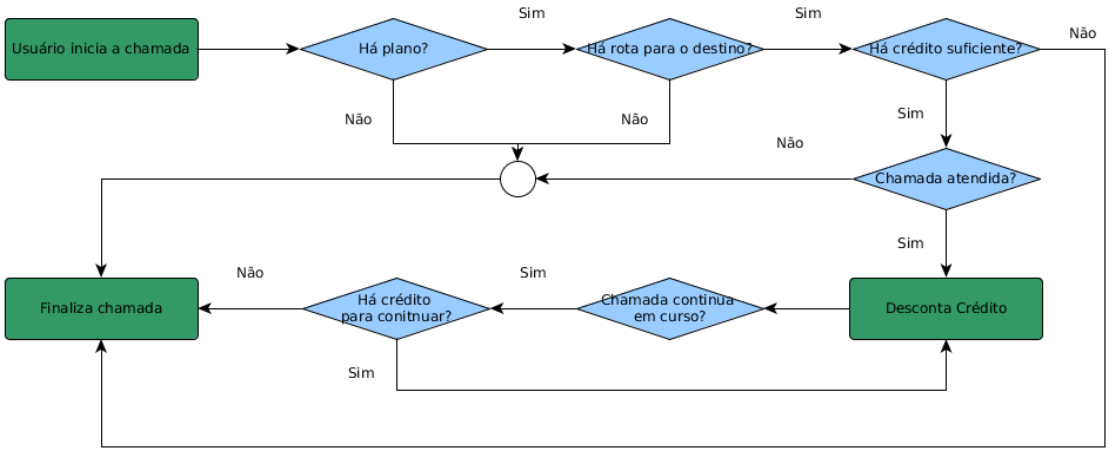

Figura 3.10: Diagrama de fluxo do tarifador

o preço equivalente a 1 minuto, mesmo que mantenha a chamada ativa por apenas 10 segundos. O 30 seria o tempo deve ser cobrado após o fim do primeiro tempo, então depois desse 1 minuto é cobrado 30 segundos e após isso é cobrado novamente mais 30 segundos no início de cada um desses 30 segundos. Então se um usuário fica 1:01 minutos no telefone, a operadora cobra 1:30.

Sabendo disso, foi encontrado o primeiro problema no desenvolvimento do tarifador utilizando o `mod_nibblebill`. A documentação dele é bem escassa e não fala sobre como é feita exatamente a tarifação. Então durante os testes, foi descoberto que ele faz a tarifação diferente do que foi descrito anteriormente. Com o `mod_nibblebill` não é possível configurar a taxa de conexão, por isso a cadência no sistema é com apenas um tempo. Ele por padrão faz desconto de créditos assim que a chamada é atendida, então caso a chamada custe 10 centavos a cada 30 segundos, ele faz o desconto de 10 centavos quando a chamada é atendida, nos próximos 30 segundos ele faz o desconto de 10 centavos novamente. Porém quando a chamada é finalizada, ele faz o desconto de mais 10 centavos.

Foi procurado algumas maneiras de contornar esse desconto que o `mod_nibblebill` faz, mas todas sem sucesso. Uma das maneiras que foi tentada inicialmente, foi a configuração das instruções SQL do módulo. Porém as variáveis que a documentação mostra que são possíveis de utilizar e que vem no arquivo de configuração padrão, não funcionaram corretamente. A documentação, por ser escassa, não mostra nada que possa ajudar nesse problema. Então foi configurado a `custom_sql_save`, que é a instrução SQL que atualiza o crédito, para atualizar o crédito de outra forma, mas não resultou em nada que fosse ajudar na tarifação.

```
1 <!-- Custom SQL for loading current balance - overrides column names
2     channel vars are interpreted.
```

```
    field nibble_balance is used for balance info

<param name="custom_sql_lookup" value="SELECT cash AS nibble_balance FROM
    accounts WHERE account_code='${nibble_account}'"/>
-->

<!-- Custom SQL for loading current balance - overrides column names
    channel vars are interpreted.
    nibble_increment is the amount to update

<param name="custom_sql_save" value="UPDATE accounts SET cash=cash-${
    nibble_increment} WHERE account_code='${nibble_account}'"/>
-->
```

Depois foi configurado dentro do XML que retorna no dialplan, inserir uma instrução que adiciona esse crédito ao usuário assim que a chamada for terminada, como no exemplo abaixo. No XML tem a configuração completa para a tarifação ser feita pelo `mod_nibblebill`. O `heartbeat` que aparece no XML é a cadência explicada anteriormente. O importante é que após a ação de "*bridge*" tem duas opções que eram uma tentativa de ajustar o crédito do usuário. Porém essa tentativa funciona parcialmente.

```
<document type="freeswitch/xml">
  <section name="dialplan" description="Exemplo Plano de Discagem">
    <context name="default">
      <extension name="1001">
        <condition field="destination_number" expression="^1001$">
     <action application="set" data="nibble_rate=0.5"/>
     <action application="set" data="nibble_account=1000"/>
     <action application="nibblebill" data="heartbeat 10"/>
     <action application="bridge" data="user/1000"/>
     <action application="set" data="nibble_adjust_credit=true"/>
     <action application="nibblebill" data="adjust 0.5"/>
        </condition>
      </extension>
    </context>
  </section>
</document>
```

O que acontece é que existem 2 conexões com o FreeSWITCH quando uma ligação ocorre, como explicado anteriormente, o conexão A que é o usuário que originou a chamada e a conexão B que é quem recebe a chamada. Esse método funciona parcialmente, pois quando B finaliza a

chamada, ele executa as opções após o *bridge*, porém quando A finaliza a chamada, a conexão entre o FreeSWITCH e A é perdida e o FreeSWITCH não continua as ações após o *bridge*. Isso é uma característica do próprio FreeSWITCH. Após algumas pesquisas, foi descoberto que existe uma opção de ação para configurar o modo zumbi para o plano de discagem. É necessário adicionar apenas `<action application="set_zombie_exec"/>` ao plano de discagem, porém isso não funcionou nos testes. É complicado encontrar algo sobre o "*set_zombie_exec*" ainda mais utilizando o `mod_nibblebill` junto, mas acredita-se que quando o conexão de A é derrubada, é derrubada também a conexão que havia com o `mod_nibblebill`, então ele perde todas referências e não executa o ajuste no final.

Também não é possível adicionar o crédito antes da chamada iniciar, pois seria necessário mudar o crédito que indica quando o crédito do usuário chega a 0 para cada chamada, o que não é possível configurar dinamicamente dentro do plano de discagem e nem no `mod_nibblebill`. Então optou-se por deixar o funcionamento por padrão, creditando mais uma vez no final da chamada.

Existe também o problema com a cadência, que teve que ser contornado de outra maneira. O `mod_nibblebill` aceita a configuração de uma cadência para cada chamada, porém ele faz cálculos em cima da cadência global configurada nele. Por exemplo, a cadência global é de 60 segundos e a cadência configurada no plano de discagem é de 10 segundos e o valor é de 0,12 centavos. Então quando a chamada é atendida deveria descontar 10 centavos e depois a cada 10 segundos descontar 10 centavos. Porém o `mod_nibblebill` faz um cálculo interno e considera que a tarifa de 0,12 centavos é referente aos 60 segundos da cadência global e não referente aos 10 segundos que foi a cadência configurada para essa chamada. Então internamente ele faz os cálculos e desconta 0,02 centavos a cada 10 segundos, que totalizando 60 segundos, fecha os 0,12 centavos. Porém contornar isso foi mais simples, só foi necessário fazer um cálculo. O cadência global, em segundos, multiplicada pelo valor que vai ser cobrado na chamada, dividido pela cadência em segundos. Isso vai dar um novo valor de cobrança que deve ser colocado na configuração da chamada. No exemplo dado, para descontar os 0,12 centavos o resultado da conta daria 0,72 centavos.

3.4 Tarifação Pós Paga

A tarifação pós paga é a tarifação que a princípio é mais simples. Como ela não precisa ser feita em tempo real, ela pode ser calculada a hora que o administrador quiser. Porém é necessário que os dados da chamada estejam devidamente guardados para poder fazer a tarifação. Essas

informações ficam armazenados no CDR, porém por padrão ele salva o CDR em um arquivo do tipo CSV (*Comma-separated values*), que é um arquivo de texto com dados separados por vírgula, regulamentado pela RFC 4180.

Inicialmente pensou-se em salvar os dados diretamente no banco de dados utilizando o `mod_cdr_pg_csv`, que é um módulo que salva os dados do CDR diretamente no PostgreSQL[1]. Porém foi decidido utilizar o MySQL antes de pensar nesse caso, então não havia tempo hábil para mudar de banco de dados, pois todo o sistema já estava implementado utilizando o MySQL.

O que foi decidido no final para conseguir importar os dados dentro do banco de dados MySQL, foi configurar o módulo padrão do FreeSWITCH para gerar um CSV com instruções de SQL. Isso é possível, pois o módulo permite a configuração de modelos de como os dados devem ser salvos. Então dentro disso, foi criado um padrão onde cada linha do CSV é uma instrução de INSERT na tabela de CDR criada para o sistema.

Após isso foi criado um *script* simples, fora da aplicação, para fazer a importação do arquivo CSV. Esse *script* é configurado para ser executado periodicamente pelo sistema operacional. O *script* inicialmente cria uma pasta para o *backup* do CDR dentro da pasta `tmp` do sistema operacional. Em seguida ele utiliza de um comando do próprio FreeSWITCH e faz uma rotação do arquivo, essa rotação renomeia o CSV atual e adiciona data e hora de quando o comando foi executado no final do arquivo. Em seguida ele cria um novo arquivo CSV em branco. Então o arquivo é inserido no banco de dados e movido para o backup.

```
1 mkdir /tmp/cdr_backup/ &> /dev/null
2 /usr/local/freeswitch/bin/fs_cli -x 'cdr_csv rotate'
3 mysql -u$DB_USER -p$DB_PASS -h$DB_IP $DB_NAME < /usr/local/freeswitch/log/cdr-
      csv/Master.csv.*
4 mv /usr/local/freeswitch/log/cdr-csv/Master.csv.* /tmp/cdr_backup/
```

Com os dados salvos no banco de dados, é possível fazer a tarifação pós-paga em cima de usuários em específico e gerar extratos. Pois como o usuário tem um plano com valores para cada rota, é necessário apenas calcular o valor da chamada em cima do tempo total dela. Com isso é gerado um CSV com o valor e dados da chamada, que pode ser utilizado para gerar as faturas ou boletos.

Foi decidido dessa forma, pois ler o arquivo em busca de um usuário apenas geraria um transtorno enorme, já que o volume de dados pode ser grande, e levando em conta que dentro do banco de dados a busca é feita de forma mais simples. Também pode ser otimizada com a utilização de índices em cima da tabela, já no arquivo isso não é viável. Na leitura de um

[1] Banco de dados relacional de código aberto.

arquivo, teria que varrer o arquivo inteiro para poder procurar os dados de um determinado usuário.

4 Conclusões

Este trabalho apresentou o desenvolvimento de um sistema de tarifação pra o FreeSWITCH. O foco principal foi a parte de tarifação pré e pós-paga.

Os objetivos impostos inicialmente foram alcançados com êxito. Porem os testes feitos durante o desenvolvimento mostrou que os caminhos tomados durante o desenvolvimento não foram os melhores. O sistema de tarifação foi feito, porém a tarifação não ocorre nem um pouco próximo das regras da ANATEL. Isso não era um objetivo do trabalho, mas era algo que eu esperava do projeto. O `mod_nibblebill` se mostrou uma opção ruim para fazer a parte de tarifação, mesmo fazendo muito bem seu trabalho, pois durante os testes ele não falhou nos valores nenhuma vez. Ele foi escolhido, pois já é um módulo do próprio FreeSWITCH e pela descrição mostra que utiliza algumas coisas interessantes, mas ele não pode ser aplicado a um tarifador comercial. Porém ele não faz a tarifação da forma que era esperada e desejada. Também tiveram algumas coisas que tiveram que ser contornadas, como por exemplo as cadências, que foi explicado na seção 2.2.3.

Como trabalhos futuros, sugere-se a reimplementação do sistema, tirando a parte do `mod_nibblebill` e fazendo por um sistema separado utilizando o *Event Socket Library* do FreeSWITCH para fazer a tarifação. Outra possibilidade é procurar a causa da tarifação diferenciada do `mod_nibblebill` no seu código fonte e alterar para a tarifação mais próxima da nossa realidade contribuindo para a comunidade como um todo.

Referências Bibliográficas

BOECHAT, E. *Sistema de Tarifação: Módulo I*. 2005. http://www.teleco.com.br/tutoriais/tutorialtarifacao/default.asp. Último acesso em 28 de Novembro de 2013.

BOTELER, J. *Mod XML Curl*. 2014. https://freeswitch.org/confluence/display/FREESWITCH/mod_xml_curl. Último acesso em 28 de Maio de 2015.

GONCALVES, F. E. *Building Telephony Systems with OpenSIPS 1.6*. [S.l.]: Packt Publishing Ltd., 2010.

JOHANSSON, O. *CDR: Call Detail Record*. 2007. http://www.voip-info.org/wiki/view/CDR. Último acesso em 30 de Julho de 2015.

MINESSALE, A. *How does FreeSWITCH compare to Asterisk?* 2008. http://www.freeswitch.org/node/117. Último acesso em 28 de Novembro de 2013.

MINESSALE, A. *FreeSWITCH 1.2*. [S.l.]: Packt Publishing Ltd., 2013.

ROUSE, M. *Softswitch Definition*. http://searchnetworking.techtarget.com/definition/softswitch. Último acesso em 30 de Julho de 2015.

SCHREIBER, D. *Mod nibblebill*. 2009. https://wiki.freeswitch.org/wiki/Mod_nibblebill. Último acesso em 27 de Maio de 2015.