UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Fábio Ricardo Boaventura Bilotto

Implementação de ferramenta de tolerância a falhas para sistema de alta disponibilidade

São Carlos

2017

Fábio Ricardo Boaventura Bilotto

Implementação de ferramenta de tolerância a falhas para sistema de alta disponibilidade

Monografia apresentada ao Curso de Engenharia Elétrica com Ênfase em Eletrônica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Dr. Carlos Dias Maciel

São Carlos

2017

Bilotto, Fabio Ricardo Boaventura
B595i Implementação de ferramenta de tolerância a falhas para sistema de alta disponibilidade / Fabio Ricardo Boaventura Bilotto; orientador Carlos Dias Maciel. São Carlos, 2017.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2017.

1. tolerância a falhas. 2. alta disponibilidade. 3. ferramenta open source. 4. failover automático. I. Título.

FOLHA DE APROVAÇÃO

Nome: Fabio Ricardo Boaventura Bilotto

Título: "Implementação de ferramenta de tolerância a falhas para sistema de alta disponibilidade"

Trabalho de Conclusão de Curso defendido e aprovado em 29 / 11 / 2017,

com NOTA 9,0 (nove, zero), **pela Comissão Julgadora:**

Prof. Associado Carlos Dias Maciel - Orientador - SEL/EESC/USP

Mestre Michel Bessani - Doutorando - SEL/EESC/USP

Mestre Tadeu Junior Gross - Doutorando - SEL/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Rogério Andrade Flauzino

Dedico este trabalho aos meus avós

AGRADECIMENTOS

Agradeço primeiramente a Deus, por sempre estar ao meu lado todos os momentos. À minha família: ao meu avô Antônio Boaventura, que nunca deixou de me apoiar em todos os sentidos. Sem ele, este trabalho jamais teria acontecido; à minha avó, Virgínia, que zela por mim desde sempre; à minha mãe, Raquel, que sempre me incentivou a ir mais longe. Aos meus tios, César, Camila e Juliana, que sempre me cobraram resultados.

Ao meu orientador, Prof. Dr. Carlos Dias Maciel, pela oportunidade de desenvolver este trabalho. Também ao doutorando Michel, pela ajuda e paciência.

A todos os professores que auxiliaram na minha formação como engenheiro e profissional.

Aos meus colegas de faculdade, por todos os momentos que passamos juntos nessa jornada, em especial: Victor, Vinícius, Pedro, Rafael, Bráulio e Francisco.

Finalmente, a todos que contribuíram de alguma forma para o desenvolvimento deste trabalho.

"Whether you think you can, or you think you can't – you're right."
Henry Ford

RESUMO

BILOTTO, F. **Implementação de ferramenta de tolerância a falhas para sistema de alta disponibilidade**. 2017. 69p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2017.

Em sistemas voltados para telefonia móvel suportado por computadores, supõem-se que o sistema opere devidamente, sem interrupção do serviço. Disponibilidade absoluta, porém, está longe de ser alcançada. A disponibilidade de serviços não é um conceito abstrato, mas sim um parâmetro de um sistema que pode ser medido quantitativamente. Podem ser utilizadas técnicas de projeto para aumentar esse número, que pode chegar bem próximo a cem por cento. Contudo, falhas são inevitáveis e sistemas infalíveis são, portanto, impossíveis de serem atingidos. É por isso que aplicam-se técnicas de tolerância a falhas em sistemas, que podem garantir o fornecimento do serviço sem interrupção mesmo em caso de falha. Este presente trabalho tem o objetivo de apresentar a implementação de uma ferramenta automática de tolerância a falhas implementada em uma empresa de telefonia móvel, bem como seus resultados.

Palavras-chave: Alta disponibilidade. Tolerância a falhas. Redundância. Ferramenta Open-Source.

ABSTRACT

BILOTTO, F. **Implementation of fault tolerance tool for high availability system**. 2017. 69p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2017.

In systems oriented to mobile telephony supported by computers, it is assumed that the system operates properly, without interruption of the service. Absolute availability, however, is far from being achieved. The availability of services is not an abstract concept, but rather a parameter of a system that can be measured quantitatively. Project techniques can be used to increase this number, which may well be close to one hundred percent. However, failures are inevitable and infallible systems are therefore impossible to achieve. This is why fault tolerance techniques are applied in systems, which can guarantee service delivery without interruption even in the event of failure. This paper aims to present the implementation of an automatic fault tolerance tool implemented in a mobile phone company, as well as its results.

Keywords: High availability. Fault Tolerance. Automatic Failover. Redundancy. Open Source Tool

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

3GPP	3*rd Generation Partnership Project*
3G	3*rd Generation*
4G	4*th Generation*
AAA	*Authentication*, *Authorization* and *Accounting*
ACK	*Acknowledgement*
AFT	Automatic Failover Tool
B2B	*Business to Business*
E-UTRAN	*Evolved Universal Terrestrial Radio Access*
EPC	*Evolved Packet Core*
ERB	Estação Rádio Base
DRA	*Diameter Routing Agent*
IMS	*IPMultimedia Subsystem*
IP	*Interner Protocol*
JSON	*JavaScript Object Notation*
LTE	*Long Term Evolution*
MTBF	*Mean Time Between Failures*
MTTR	*Mean Time To Repair*
P2P	*Peer to Peer*
PCC	*Policy Control and Charging*
PCEF	*Policy and Charging Enforcement Function*
PCRF	*Policy and Charging Rules Function*
REST	Representation State Transfer
SMNP	*Simple Network Management Protocol*
SMS	*Short Message Service*

SOAP	Simple Object Access Protocol
SPR	*Subscriber Profile Repository*
SCTP	*Stream Control Transmission Protocol*
TCP	*Transmission Control Protocol*
TPS	Transações por segundo
VM	*Virtual Machine*
W3C	*World Wide Web Consortium*
WWW	*World Wide Web*
XML	Extensible Markup Language

LISTA DE SÍMBOLOS

G_x	interface G_x
t_d	tempo disponível
t_i	tempo indisponível
d	disponibilidade

SUMÁRIO

1 INTRODUÇÃO

A necessidade de alta disponibilidade não teve origem recentemente com o avanço da Internet ou o comércio virtual. Na verdade, esse conceito existe há milhares de anos. Quando navios gregos partiam para descobrir novas terras, os capitães faziam questão de levar a bordo velas e remos de reposição. Se algum desses itens falhasse, a equipe imediatamente o substituiria e continuava a viagem, enquanto reparavam o danificado (JAYASWAL, 2005).

Com a nossa crescente dependência de sistemas de informação, a alta disponibilidade assumiu um novo significado e importância. Empresas e consumidores estão se voltando para a Internet para comprar bens e serviços. Pessoas realizam negócios a qualquer momento a partir do computador, ou melhor, do celular.

A indústria de telecomunicações também está em um período de grandes mudanças com a rápida convergência dos serviços para dispositivos móveis. Atualmente, os telefones móveis são considerados computadores pessoais, com alto poder de processamento e armazenamento de dados, não mais sendo utilizados apenas para chamadas de voz (MAGUELA; AQUINO, 2015).

As redes de telefonia também precisaram se adaptar para passar a oferecer serviços cada vez mais rápidos e confiáveis. As redes de telefonia de quarta geração (4G) foram instaladas com o intuito de suprir a necessidade dos usuários móveis por serviços de alta taxa de transmissão. O padrão conhecido como LTE (Long Term Evolution) se tornou a principal rede móvel 4G para transmissão de dados (MAGUELA; AQUINO, 2015). Algumas provedoras de telefonia já prometeram até o fim do ano corrente a instalação de redes 4.5G. Fora do país, já se fala muito em redes 5G.

A exigência dos consumidores tem feito as companhias de telefonia móvel buscarem por sistemas cada vez melhores e mais capazes de atendê-los. Não apenas isso, mas também o uso da rede de dados para serviços B2B (*Business to business*) está em alta. Sistemas de alta disponibilidade tornam-se cada vez mais essenciais em aplicações no ramo de telecomunicações. Isso porque paradas indesejadas neste tipo de sistema estão intimamente associadas à significantes perdas de receita para a companhia, além de que afetam diretamente os clientes finais, prejudicando a reputação e imagem da empresa.

De acordo com Weber (WEBER, 2003), falhas são inevitáveis, mas as consequências destas, como por exemplo a interrupção no fornecimento do serviço, podem ser evitados pelo uso de técnicas de fácil compreensão. Para estas empresas, portanto, um plano anti-falhas é fundamental para agregar qualidade ao serviço e evitar interrupções. Além disso, a concorrência de mercado e a exigência dos consumidores tem incitado as empresas

a minimizarem taxas de falhas (SCHMIDT, 2017).

É nesse contexto que foi desenvolvida uma ferramenta de tolerância a falhas para uma empresa que atua no ramo de telecomunicações. A companhia já aplicava diversos conceitos de alta disponibilidade em seus sistemas, como a redundância, mas quando um cliente requisitou uma ferramenta automática de tolerância a falhas, o trabalho que deu origem a esta monografia teve início.

1.1 Objetivos

O presente trabalho tem como objetivo apresentar ao leitor a implementação de uma ferramenta de tolerância a falhas projetada para melhorar a disponibilidade de um sistema.

Primeiramente serão introduzidos os conceitos de alta disponibilidade e suas métricas. Também será descrito o sistema LTE, bem como seus componentes e suas padronizações.

Então, será descrito o sistema em que a ferramenta de tolerância a falhas foi aplicado, antes de finalmente ser apresentada a ferramenta em si.

O trabalho foi estruturado de maneira que o leitor não precise possuir nenhum conhecimento prévio sobre o assunto. Os conceitos envolvidos no trabalho são descritos de forma gradual, de forma que a leitura não se torne cansativa.

1.2 Estrutura do trabalho

O trabalho foi estruturado em seis capítulos, que são:

- Capítulo 1: apresenta a introdução e os objetivos do trabalho;
- Capítulo 2: apresenta o conceito de alta disponibilidade e conceitos relacionados;
- Capítulo 3: descreve o principal sistema de telecomunicações, bem como seus componentes e suas padronizações;
- Capítulo 4: descreve o sistema em que foi implementada a ferramenta de tolerância a falhas;
- Capítulo 5: apresenta a ferramenta de tolerância a falhas;
- Capítulo 6: apresenta os resultados da implementação da ferramenta;
- Capítulo 7: apresenta as conclusões do trabalho.

2 ALTA DISPONIBILIDADE

Com a crescente dependência de sistemas computacionais para diversos tipos de atividades, torna-se uma preocupação manter esses sistemas disponíveis, se possível, todo o tempo. Sistemas com disponibilidade contínua são ideais, porem teóricos e hipotéticos.

Em termos técnicos, disponibilidade é a probabilidade de que um sistema estará operacionalmente disponível em dado momento. Alta disponibilidade pode ser encontrada em sistemas que garantem soluções em torno de 99,99% a 99,9999% de disponibilidade. Existe competição entre as companhias para ver quem consegue adicionar mais noves (MOSER, 2004).

De acordo com Costa (COSTA, 2009), o objetivo de promover alta disponibilidade resume-se na garantia de que um serviço esteja sempre disponível quando um cliente ou usuário requisitá-lo. Já Batista (BATISTA, 2007), diz que o objetivo da alta disponibilidade é manter os serviços prestados por um sistema mesmo que este venha a falhar.

2.1 Tolerância a falhas

Segundo Jayaswal, tolerância a falhas é a capacidade de um sistema se recuperar de uma falha de um componente sem interrupção do serviço (JAYASWAL, 2005). Já Avizienis (AVIZIENIS, 1998) diz que um sistema é tolerante a falhas se suas funções podem ser executadas independentemente da ocorrência de falhas.

Gartner (GARTNER, 1967) diz que sobre sistemas tolerantes a falhas, duas coisas são certas. Primeiramente, não importa quão bem projetado um sistema é ou de quão boa qualidade são seus componentes, sempre existe a possibilidade de ele falhar, pois falhas são inevitáveis e podem acontecer por qualquer motivo. Falhas comuns em sistemas de alta disponibilidade são devido a condições meteorológicas, mas elas também podem ocorrer por simples falhas de equipamento ou cortes acidentais de energia elétrica. Não importa a razão, uma interrupção no serviço pode custar a uma companhia muito dinheiro.

Em segundo lugar, Garter demonstrou que, para atingir alta disponibilidade ou tolerância a falhas, é imprescindível a aplicação de alguma forma de redundância. Redundância é a principal técnica utilizada para aumentar a disponibilidade de um sistema. É praticamente impossível falar de alta disponibilidade sem falar de redundância.

Redundância

Uma definição típica de redundância em engenharia, segundo Allen (ALLEN, 2017), é a duplicação de componentes críticos ou funções em um sistema com a intenção de aumentar a confiabilidade do sistema. Segundo Marshall (MARSHALL; CHAPMAN,

2002), ao aplicar redundância, circuitos alternativos, equipamentos ou componentes são instalados de modo que, em caso de falha, a funcionalidade seja preservada.

Redundância pode ser aplicada de diversas maneiras, mas duas arquiteturas se destacam no trabalho:

Redundância 1+1

A configuração 1+1 é um exemplo de redundância em paralelo. Nessa, dois sistemas equivalentes e independentes são conectados em paralelo. Os dois sistemas podem desempenhar juntos uma mesma função (Ativo-Ativo), dividindo a carga entre eles, ou podem ser configurados como Ativo-Passivo, em que um sistema desempenha as funções sozinho, enquanto o outro fica em espera, pronto para assumir caso o primeiro venha a falhar.

Redundância geográfica pode ser aplicada nesses sistemas para garantir ainda mais disponibilidade. Isso significa que cada sistema está geograficamente isolado do outro, podendo estar sujeitos a condições meteorológicas diferentes e evitando falhas do tipo desastre natural.

Redundância N+1

A configuração N+1 é capaz de assegurar a disponibilidade do sistema em caso de falha de um componente. O mínimo de componentes necessários (N) possui um componente extra de suporte. Esse componente pode ser ativo, trabalhando juntamente com os outros componentes para entregar o serviço, ou passivo, ou seja, em espera pronto para assumir caso um componente venha a falhar.

Segundo Marshall (MARSHALL; CHAPMAN, 2002), a redundância N+1 é uma alternativa de menor custo do que a redundância 1+1 e uma forma de garantir mais disponibilidade através da disponibilização de uma unidade redundante extra.

2.2 Tempo indisponível

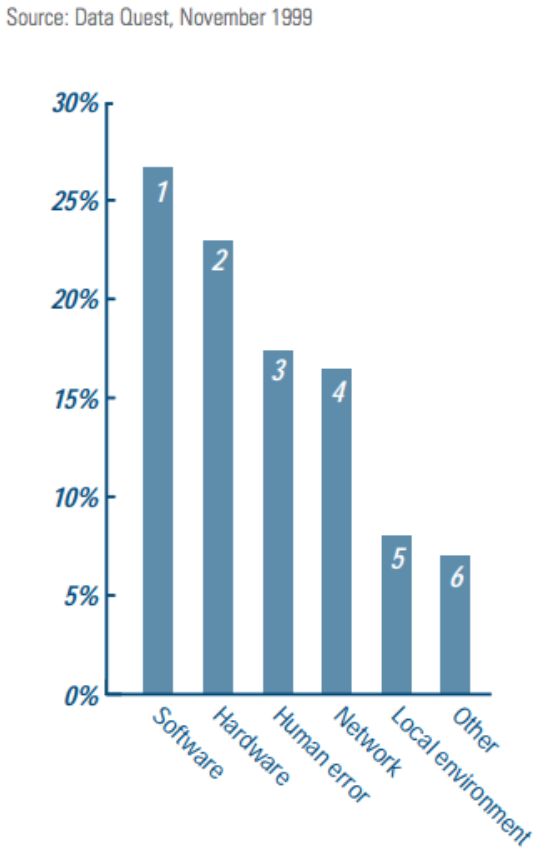
O objetivo de qualquer solução de alta disponibilidade é ter capacidade de se recuperar de uma falha dentro do menor tempo possível, causando pouca ou nenhuma janela de tempo indisponível. Conforme um dos artigos da CNT sobre alta disponibilidade (CNT, 2003), tempo indisponível se refere a uma interrupção do serviço em qualquer camada de um ambiente de aplicação. Ambiente de aplicação se refere não somente a todo *hardware* e *software* requerido para prover uma função, mas também inclui os recursos humanos envolvidos e seus ambientes de trabalho.

Causas

Diversos fatores podem causar o *downtime* ou tempo indisponível de uma aplicação. *Software* pode apresentar *bugs* que necessitam ser corrigidos. O *hardware* pode falhar, por mais confiável que seja. Esses fatores podem requerer manutenção ou *downtime* programado. Indisponibilidade também pode ser causada por outras áreas da empresa, como por exemplo a área responsável pela manutenção da energia elétrica. Além disso, há problemas externos que estão fora do alcance de todos, como por exemplo desastres naturais (BATISTA, 2007).

Um estudo de 1999, da revista *Data Quest* (CNT, 2003), mostrou que apenas 23 por cento das interrupções dos serviços eram causadas por falhas no *hardware*. O estudo mostrou que 27 por cento era causado por falhas no *software*. Outras fontes de indisponibilidade incluem falhas humanas (18 por cento). A Figura 1 demonstra os resultados deste estudo.

Figura 1: Causas comuns de falha



Fonte: (CNT, 2003)

Custos

Por custo de indisponibilidade, entende-se como sendo as consequências ou os prejuízos que uma empresa pode ter em decorrência da interrupção da aplicação (BATISTA, 2007).

O custo de indisponibilidade varia muito de indústria para indústria. Um estudado publicado pelo Meta Group em 2000 (CNT, 2003), coloca os custos de *downtime* para indústrias comuns entre 500 mil até três milhões de dólares por hora. A Figura 2 demonstra os resultados desse estudo.

Figura 2: Custo da indisponibilidade em diversos tipos de indústria

Source: Meta Group, Individual.com, October 2000

Industry	$ per hour
Energy	$3M
Telecommunications	$3M
Finance	$1.5M
IT-dependent mfg.	$1.5M
Healthcare	$0.5M
Media	$0.5M
Hospitality/Travel	$0.5M

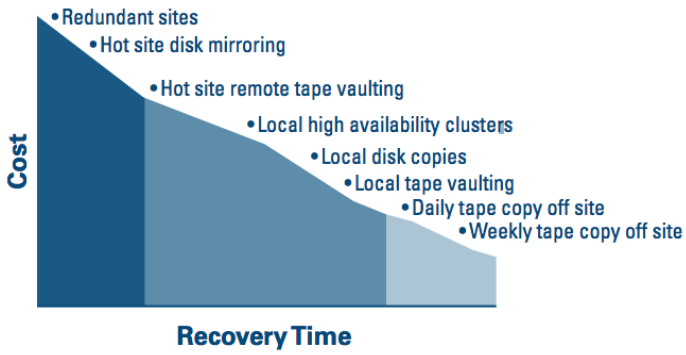
Fonte: (CNT, 2003)

Quantificar o custo da indisponibilidade é útil pois detalha de maneira clara o risco que uma companhia enfrenta. Uma maneira de mitigar este risco é através de soluções de alta disponibilidade. A Figura 3 demonstra os custos de implementação de sistemas disponíveis. Nota-se que sistemas redundantes com pouco tempo de indisponibilidade são os mais caros.

2.3 Medição de disponibilidade

Não há como afirmar que uma solução é mais disponível que outra ou que esta melhorou a sua disponibilidade se não há como medi-la. A disponibilidade de um sistema é a probabilidade de encontrá-lo operando em determinado momento. Essa probabilidade, portanto, leva em conta o *uptime* (tempo em que o sistema está operando) e o *downtime* (FILHO, 2004).

Figura 3: Custo da disponibilidade em função do tempo de recuperação



Fonte: (CNT, 2003)

Uma notação comum para representar disponibilidade é a partir do "número de noves". Dessa forma, uma solução que apresenta cinco noves de disponibilidade está disponível por 99,999% do tempo. Ter cinco noves de disponibilidade é praticamente impossível sem algum tipo de redundância total (BATISTA, 2007).

A Tabela 1 mostra alguns exemplos de sistemas conhecidos, bem como suas disponibilidades.

Tabela 1: Exemplos de sistemas e suas disponibilidades

Disponibilidade	**Tempo indisponível por ano**	**Exemplo**
90%	35,5 dias	computadores pessoais
98%	7,3 dias	
99%	3,65 dias	sistemas de acesso
99,8%	17 horas e 30 minutos	
99,9%	8 horas e 45 minutos	provedores de Internet
99,99%	52,5 minutos	CPD, sistemas de negócios
99,999%	5,25 minutos	sistemas de telefonia; sistemas de saúde; sistemas bancários
99,9999%	31,5 segundos	sistemas de defesa militar

Fonte: (FILHO, 2004)

A disponibilidade é calculada através da Equação 2.1, em que MTBF corresponde ao *Mean Time Between Failures*, ou seja, o tempo médio entre as falhas de um serviço e MTTR *Mean Time to Repair* representa o tempo médio para reparar o serviço. Esses parâmetros e equação são provenientes de modelos probabilísticos estudados em engenharia de confiabilidade.

$$d = \frac{MTBF}{MTBF + MTTR} \quad x \quad 100\% \tag{2.1}$$

Tendo em posse apenas os tempos em que um sistema esteve *up* ou *down*, a disponibilidade pode ser calculada pela Equação 2.2. O resultado de ambas as equações é a disponibilidade do sistema em porcentagem.

$$d = \frac{T_{up}}{T_{up} + T_{down}} \quad x \quad 100\% \tag{2.2}$$

3 O SISTEMA LTE

De acordo com Schmidt (SCHMIDT, 2017), um sistema é um conjunto de componentes arranjados de acordo com um projeto específico com a finalidade de atender determinadas funções com desempenho e confiabilidade adequados. O sistema tratado neste Capítulo, conhecido como sistema LTE, é o sistema que atualmente entrega voz e serviço de dados para 1,3 bilhões de pessoas e espera entregar para até 3,8 bilhões em 2020 (SKINNER, 2016).

LTE é baseada em padrões criados pela 3^{rd} *Generation Partnership Project* (3GPP). O 3GPP é o órgão do ramo de telecomunicações que padroniza e define a rede de telefonia móvel. Esta organização escreve as definições, os protocolos utilizados entre os sistemas, os diferentes blocos estruturais, como estes se comunicam entre si e como o sistema todo deve funcionar. De acordo com o site da organização, o projeto abrange as tecnologias de redes de telecomunicação celular e fornece as especificações completas do sistema.

De acordo com (OLSSON et al., 2013), a necessidade de uma padronização global para a rede móvel é direcionada por muitos fatores, mas principalmente para garantir a interoperabilidade entre os sistemas em um verdadeiro meio de multi-vendedores. Operadores precisam ter certeza que eles poderão comprar dispositivos de rede de várias companhias, aumentando a conpetição entre estas. Para que isso seja possível, os aparelhos de diferentes vendedores devem trabalhar uns com os outros e isto só pode ser atingido especificando uma série de descrições de interface, em que os diferentes nós são capazes de comunicar-se uns com os outros.

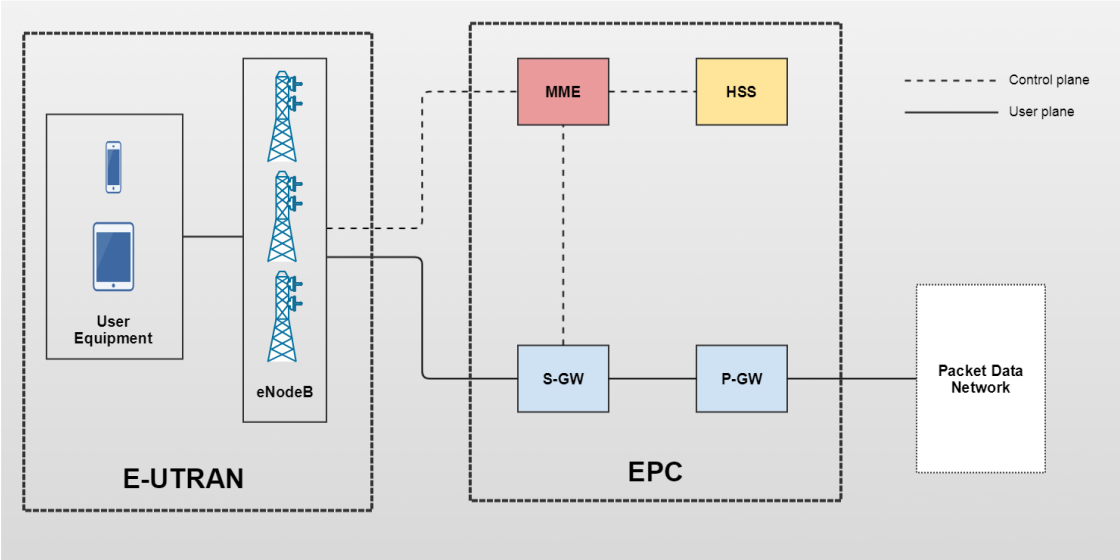
3.1 O *Evolved Packet System*

O sistema LTE é formado por uma rede de acesso chamada E-UTRAN (*Evolved Universal Terrestrial Radio Access*) e um núcleo de rede conhecido como EPC (*Evolved Packet Core*). Combinados, o EPC, E-UTRAN e o *User Equipment* (UE) ou dispositivo móvel (celulares, *tablets*), formam o EPS (Evolved Packet System).

A Figura 4 mostra uma arquitetura básica do EPS quando o equipamento do usuário (UE) está conectada ao EPC através de E-UTRAN, O *Evolved NodeB* (eNodeB) é a estação radio base LTE. Estações Radio Base (ERB) são equipamentos que fazem a conexão entre os telefones celulares e a companhia telefônica. Na figura o EPC é composto de quatro elementos de rede: o *Serving Gateway* (S-GW), o *PDN Gateway* (P-GW), o *Mobility Management Entity* (MME) e o *Home Subscriber Server* (HSS).

Existe pelo menos um eNodeB em uma rede de rádio LTE. Na verdade, em uma rede de tamanho razoável, podem existir milhares. Todos as estações rádio base são

Figura 4: Arquitetura básica EPS com acesso E-UTRAN



Fonte: Firmin (2017)

conectadas em pelo menos um MME. O MME é o elemento responsável pela autenticação e autorização de usuários na rede. Ele lida com a sinalização relacionada à mobilidade e segurança para o acesso E-UTRAN. O HSS, por sua vez, é a *database* que contém informações relacionadas aos usuários e assinantes. O HSS é responsável por armazenar as informações de identificação e autenticação dos usuários da rede LTE (MAGUELA; AQUINO, 2015).

Os *gateways* (S-GW e P-GW) lidam com o o plano do usuário. Eles transportam o tráfego de dados IP entre o UE (User Equipment) ou equipamento do usuário e as redes externas. O 3GPP especifica os *gateways* como entidades independentes, mas na prática eles podem ser combinados em uma única "caixa" pelos vendedores de equipamentos de rede (FIRMIN, 2017). O S-GW é o ponto de conexão entre o E-UTRAN e o EPC. Tal como seu nome, este *gateway* serve o equipamento do usuário através do roteamento dos *packets*, mais especificamente *IP packets*, que chegam e vão. O P-GW é o ponto de conexão entre o EPC e as redes IP externas. Essas redes são conhecidas como PDN (*Packet Data Network*).

PDN é uma descrição genérica para uma rede que fornece serviços de dados por meio de *packets*. Um *packet* é uma estrutura unitária formatada de dados transportada por uma rede que utilize comutação de pacotes. *Packet switching* ou Comutação de Pacotes é um modo de transmitir dados em que as mensagens são divididas em diversas partes que são enviadas de forma independente e remontadas no destino final. A Internet é um exemplo de PDN (DIALOGIC, 2017).

O P-GW é responsável por rotear *packets* do EPC para o PDN e do PDN para o

EPC. Além disso, o P-GW tem um papel chave no conceito de *policy control and charging* ou controle de políticas e cobrança. Este conceito também é definido e padronizado pelo 3GPP como um subsistema do EPS conhecido como *PCC Architecture*.

Policy Control and Charging é um conceito fundamental na rede LTE que está relacionado à cobrança baseada em quantidade de dados utilizados e também ao controle de políticas.

3.2 PCC *Architecture*

Definido na especificação 23.203 (3GPP-23.203, 2017), o *framework* PCC é uma arquitetura que permite o controle de políticas na rede LTE e também está relacionado a cobrança dos assinantes por parte das operadoras.

Uma política, na arquitetura 3GPP, é uma regra ou um tratamento que um IP específico recebe na rede, como por exemplo, como os dados serão cobrados ou qual *Quality of Service* (QoS) receberá um serviço (OLSSON et al., 2013).

O QoS a que é submetido o assinante é o valor do *bit rate* em que ele pode fazer *download* e *upload*. Bit rate é o número de bits convertidos ou processados por unidade de tempo. O bit rate é medido em 'bits por segundo' (bps ou b/s), muitas vezes utilizado em conjunto com um prefixo SI, como kbps ou Mbps. Sem grande rigor, QoS é a velocidade da internet de um assinante.

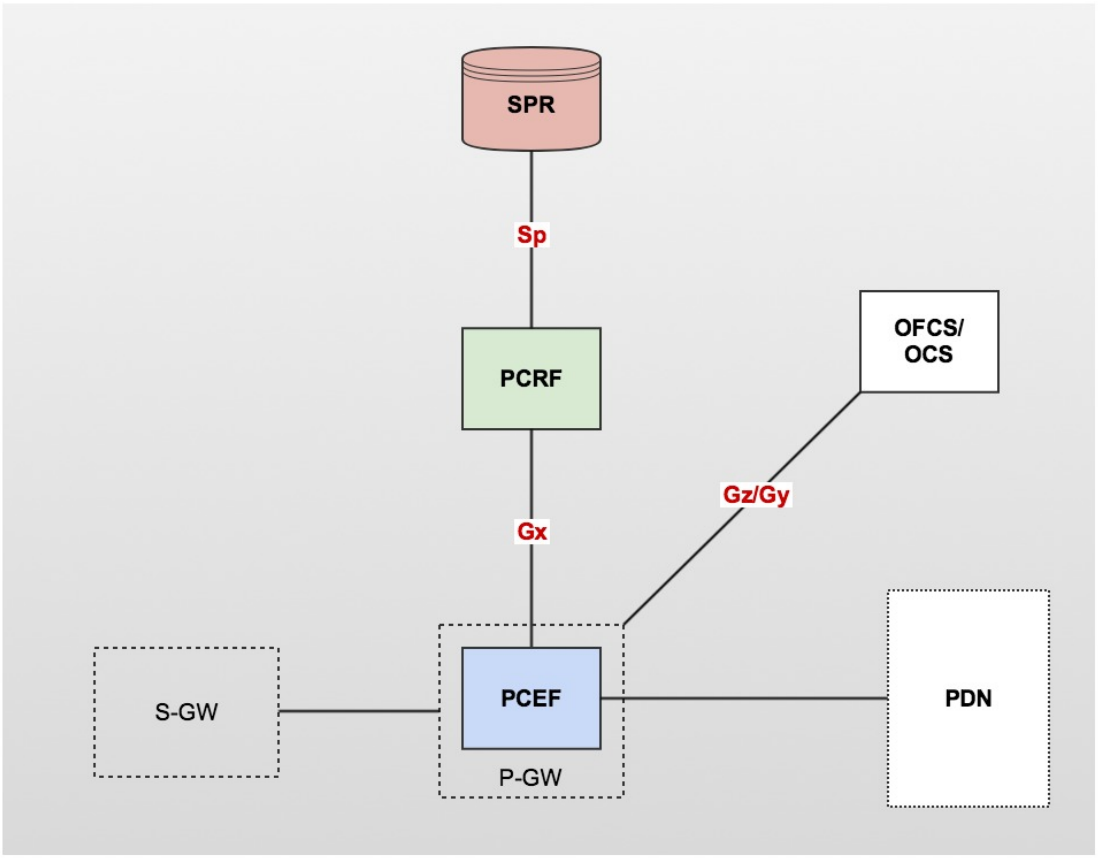
O PCC é representado na Figura 5. Nem todas as entidades e interfaces entre os elementos do PCC foram representadas na figura por não fazerem parte do escopo deste trabalho. O leitor é convidado a checar as referências para aprofundar-se na arquitetura PCC, seus elementos e interfaces.

3.2.1 *Policy and Charging Rules Function*

O PCRF (*Policy and Charging Rules Function*) é o componente chave responsável pelo controle de políticas da arquitetura PCC. Ele tem o papel fundamental de determinar as regras de navegação aos usuários da rede LTE, definindo se e quando determinado assinante tem permissão para navegar, com qual QoS e qual quantidade de uso. O PCRF também define quais serviços serão monitorados para cada usuário. Por exemplo, algumas operadoras atualmente fornecem serviços grátis, como *Facebook* e *Whatsapp*. É função do PCRF definir que esses serviços não serão cobrados.

O PCRF está estritamente relacionado ao PCEF (*Policy and Charging Enforcement Function*), que garante que as políticas definidas por aquele sejam impostas na rede para cada usuário. O PCRF fornece controle de rede baseado em detecção de fluxo de dados e QoS para o PCEF (3GPP-29.214, 2017), enquanto este basicamente lida com o tráfego

Figura 5: Arquitetura PCC



Fonte: 3GPP-23.203 (2017)

do usuário. O PCEF reporta ao PCRF quando um usuário conecta-se na rede, quando o mesmo atinge a sua quantidade de uso fornecida por este e quando o usuário desconecta-se.

O PCEF também reporta ao PCRF quando há uma mudança na tecnologia de acesso. Por exemplo, quando um usuário sai da área de cobertura 4G e entra na área 3G. O PCEF reporta o ocorrido ao PCRF, que alterará a velocidade do usuário. Além disso, quando um usuário viaja para outro país e conecta-se na rede local (*roaming*), o PCEF local identifica o caso e reporta ao PCRF local, que provavelmente aplicará um QoS baixo para esse usuário.

Em todos os casos, há uma resposta ou ACK (*acknowledgement*) do PCRF para o PCEF. ACK é um sinal transmitido entre processos de comunicação que significa reconhecimento, ou uma confirmação do recebimento da mensagem, como parte de um protocolo de comunicação.

Vale ressaltar que a comunicação entre o PCRF e o PCEF se dá através da interface *Gx* e é padronizada pelo protocolo *Diameter*.

Protocolo *Diameter*

Diameter é um protocolo de rede que oferece serviços de AAA (*Authentication*, *Authorization* e *Accounting*) para aplicações que envolvam acesso a rede móvel. De acordo com Souza, Pires e Lima (2015), ele é amplamente utilizado em redes que transportam informação e serviços de voz, dados, entre outros, através de pacotes, como as redes IMS (IPMultimedia Subsystem) e o próprio EPS.

O DIAMETER usa tanto o *Transmission Control Protocol* (TCP) quanto o *Stream Control Transmission Protocol* (STCP) para transporte de mensagens entre os *peers*. O TCP e STCP exigem que seja estabelecida uma conexão antes de qualquer mensagem DIAMETER ser enviada (SOUZA; PIRES; LIMA, 2015).

A Tabela 2 mostra algumas das mensagens padronizadas pelo *Diameter* que são trocadas entre o PCRF e o PCEF. A título de exemplo, quando um usuário inicia uma sessão na rede, o PCEF envia um CCR_I (I de inicial) ao PCRF, que responde com um CCA_I. Por exemplo, quando um usuário atinge sua cota determinada pelo PCRF, o PCEF envia um CCR_U (U de *update*) e é respondido com um CCA_U.

Tabela 2: Alguns comandos comuns *Diameter* definidos no protocolo

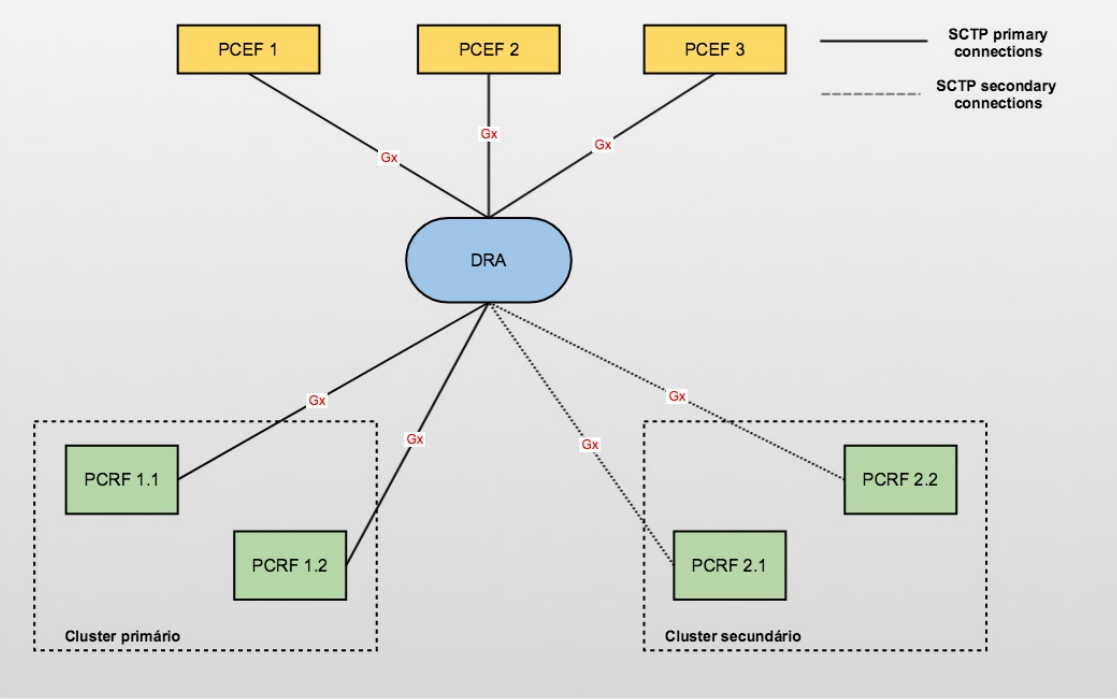
Nome do comando	Abreviatura	Código
Capabilities-Exchange-Request	CER	257
Capabilities-Exchange-Answer	CEA	257
Credit-Control-Request	CCR	272
Credit-Control-Answer	CCA	272
Device-Watchdog-Request	DWR	280
Device-Watchdog-Answer	DWA	280

Diameter Routing Agent

Com o crescente tráfego de dados móveis, tornou-se indispensável o uso do DRA (Diameter Routing Agent) em sistemas que utilizam procotolo *Diameter*. O DRA foi introduzido pelo 3GPP para lidar com o aumento do tráfego de sinalização Diameter e a crescente complexidade das redes LTE (F5, 2017). O DRA é um elemento funcional que garante que todas as sessões *Diameter* estabelecidas nos pontos de referência G_x para um determinado usuário atinjam o mesmo PCRF quando múltiplos PCRF são implementados em um sistema *Diameter*. O DRA não é necessário em uma rede que implante um único PCRF (3GPP-29.213, 2017). Portanto, o DRA fornece roteamento em tempo real, garantindo que mensagens da rede sejam transmitidas entre os elementos corretos e que as conexões apropriadas sejam estabelecidas. Vale ressaltar que as conexões estabelecidas entre o DRA e o PCRF e PCEF são do tipo P2P (*Peer to Peer*) segundo o protocolo SCTP (*Stream Control Transmission Protocol*).

Situando-se entre o PCRF e o PCEF, o DRA atua como um *load balancer*. *Load balancing* ou balanceamento de carga melhora a distribuição de de carga através de múltiplos recursos computacionais, como computadores e *cluster*. O balanceamento de carga visa otimizar o uso de recursos, maximizar a taxa de transferência, minimizar o tempo de resposta e evitar a sobrecarga de qualquer recurso. Segundo Teodoro (TEODORO et al., 2004), balanceamento de carga é um mecanismo usado para atingir escalabilidade.

Figura 6: DRA - o ponto comum entre o PCRF e o PCEF



O DRA e o conceito de *load balancing* são muito importantes no processo de *failover* pois o DRA é o único ponto comum entre o PCRF e o PCEF. A Figura 6 ilustra esse conceito. O *cluster* ativo e passivo de PCRF possuem conexões SCTP ativas com o *peer* do DRA, mas este prioriza as conexões do *cluster* ativo. Caso o *cluster* ativo falhe, o DRA imediatamente começa a direcionar o tráfego para o *cluster* passivo, não havendo necessidade de reconfigurar todos o *cluster* de PCEF.

3.2.2 *Policy and Charging Enforcement Function*

O PCEF é o elemento funcional que engloba a execução de políticas determinadas pelo PCRF e funcionalidades de cobrança. Localizada no P-GW, essa entidade tem controle sobre o tráfego do usuário e seu QoS e provê detecção e contagem do fluxo de dados, assim como interações com sistemas de cobrança *online* e *offline* (BARTON, 2012).

O PCEF deve permitir que o serviço de dados passe através do *Gateway* se e somente se houver uma *PCC rule* definida pelo PCRF. Quando requisitado, o PCEF deve reportar ao PCRF mudanças relacionadas ao serviço de dados, como por exemplo, uma

alteração na localização do usuário (o usuário pode entrar em uma região que só há acesso 3G, por exemplo).

O PCEF é logicamente conectado ao OFCS (*Offline Charging System*) e OCS (*Online Charging System*) através das interfaces Gz e Gy. Esses sistemas permitem que a operadora de telefonia cobrem seus clientes baseado no uso do serviço. O OCS, inclusive, é utilizado para cobranças em tempo real.

3.2.3 ***Subscriber Profile Repository***

A entidade lógica SPR contém informações relacionadas aos assinantes e assinaturas necessárias para tomada de decisões de políticas (3GPP-23.203, 2017) e definição das *PCC rules* por parte do PCRF. O SPR é a base de dados que foi originalmente definida para manter dados de assinaturas para o *framework* PCC. Comparado ao HSS, o SPR armazena regras mais dinâmicas e comerciais necessárias para o PCC, enquanto o HSS contém dados de assinaturas mais estáticos necessários para acesso a rede (OLSSON et al., 2013).

Já foi visto que quando o PCEF estabelece uma sessão na rede, ele envia a mensagem *Diameter* CCR_I para o PCRF. Este então, primeiramente consulta o perfil do assinante, que inclui suas características e serviços contratados no SPR para então definir uma regra para este usuário. O PCRF consulta informações no SPR como quantos dados esse usuário já consumiu no ciclo de seu plano e qual qualidade de serviço o seu plano oferece. O PCRF então, define uma *PCC rule* para este usuário e envia ao PCEF através do CCA_I, que executa essa regra nesse usuário.

O PCRF também pode estar constantemente atualizando as informações no SPR com base nas informações oriundas do PCEF. O PCEF informa ao PCRF quando determinado usuário atingiu a cota de dados que este determinou. O PCRF então, incrementa esse valor gasto na base de dados, e checa se ele ainda possui dados para serem consumidos no seu plano. Se sim, o PCRF garante mais dados ao usuário, se não, pode tomar outras decisões como o bloqueio da internet ou redução do QoS. Tudo dependerá das configurações pré-determinadas pela operadora, tais quais plano de dados, área de cobertura, etc.

O ponto de referência Sp situa-se entre o SPR e o PCRF. Segundo 3GPP-23.203 (2017), a interface Sp permite que o PCRF requisite informação relacionada ao assinante e suas características relevantes e assim possa aplicar políticas a um assinante.

4 IMPLEMENTAÇÃO DE UM SISTEMA PCC

O EPC e o PCC envolvem diversas entidades padronizadas pelo *3GPP*, mas cabe a cada projetista definir qual será a configuração do sistema a ser implementado para operadoras de telefonia móvel. Cada uma delas possui suas particularidades, como a quantidade de clientes, quantidade de ofertas oferecidas ao assinante, área de cobertura, etc.

Para o projeto descrito neste capítulo, foi requisitado um sistema de PCRF com suporte para integração com diversos outros módulos já pré-existentes no sistema. O cliente já possuía previamente um PCRF provido por outra companhia, mas quando optou por trocar para a solução de outra empresa, o sistema foi reconfigurado da maneira descrita nas próximas seções.

4.1 Definição de componentes

A partir de informações providas pelo cliente sobre sistemas anteriores, estimou-se a quantidade e particularidades do *hardware* que será utilizado. Informações como quantas sessões ativas e concorrentes são mantidas e quantas transações deverão ser processadas ou se há ou não a necessidade de arquivos serem lidos e escritos são fundamentais no projeto desse tipo de sistema. Quantidades de processos maiores implicam em mais dados para serem processados, mas esse critério está intimamente relacionado ao tempo em que esses dados necessitam ser processados. Latência baixa é preferível e obrigatória em alguns processos e sistemas.

Sistemas de telecomunicações normalmente exigem diversos computadores para lidar com o alto tráfego de dados e resolver tarefas que apenas um não conseguiria. Um *cluster* consiste em dois ou mais computadores que trabalham juntos para prover uma solução (HOCHSTETLER; BERINGER, 2004). Ainda segundo Hochstetler, um *cluster* de alta disponibilidade é tipicamente construído com a intenção de prover um ambiente seguro contra falhas através da redundância, ou seja, prover um ambiente computacional em que a falha de um ou mais componentes, não seja significante para afetar a disponibilidade da aplicação em uso.

Foi definido que cada hardware neste projeto seria separado em duas máquinas virtuais (VM, do inglês virtual machines) distintas. Ao separar o hardware em VMs distintas, garante-se que os recursos da máquina como CPU e memória não serão concorrentes entre os processos executados em cada VM. É estratégia da companhia adotar máquinas virtuais separadas para máquinas voltadas para base de dados e máquinas de aplicação. Em termos de performance e organização lógica essa prática é eficiente, pois os recursos de base de

dados não serão concorrentes com os recursos de aplicação. Aplicação é um termo genérico utilizado para referir-se as máquinas virtuais que rodam o componente PCRF. Outros componentes responsáveis por outros processos também podem ser encontrados na VM de aplicação.

A Figura 7 representa um *hardware* do sistema. Nota-se que a máquina de aplicação é representada apenas com componente PCRF, mesmo que isso não seja obrigatório. A VM de aplicação também pode possuir outros componentes responsáveis por outros processos. As máquinas de *database*, por outro lado, rodam os componentes Dados e Sessões, responsáveis pelo armazenamento de dados dos usuários (SPR) e das sessões Gx, respectivamente.

Figura 7: Representação de um *hardware* do projeto separado em duas máquinas virtuais distintas

Definiu-se para o sistema que duas máquinas de aplicação e base de dados seriam suficientes para cumprir todos os requisitos de TPS (Transações por segundo) e latência do sistema. A estratégia adotada pela companhia é utilizar-se de um *cluster* local com redundância N+1. Nesse caso, um *cluster* local composto de três nós. A fim de garantir mais disponibilidade, outro grupo de máquinas idêntico geograficamente isolado é utilizado como redundância em paralelo.

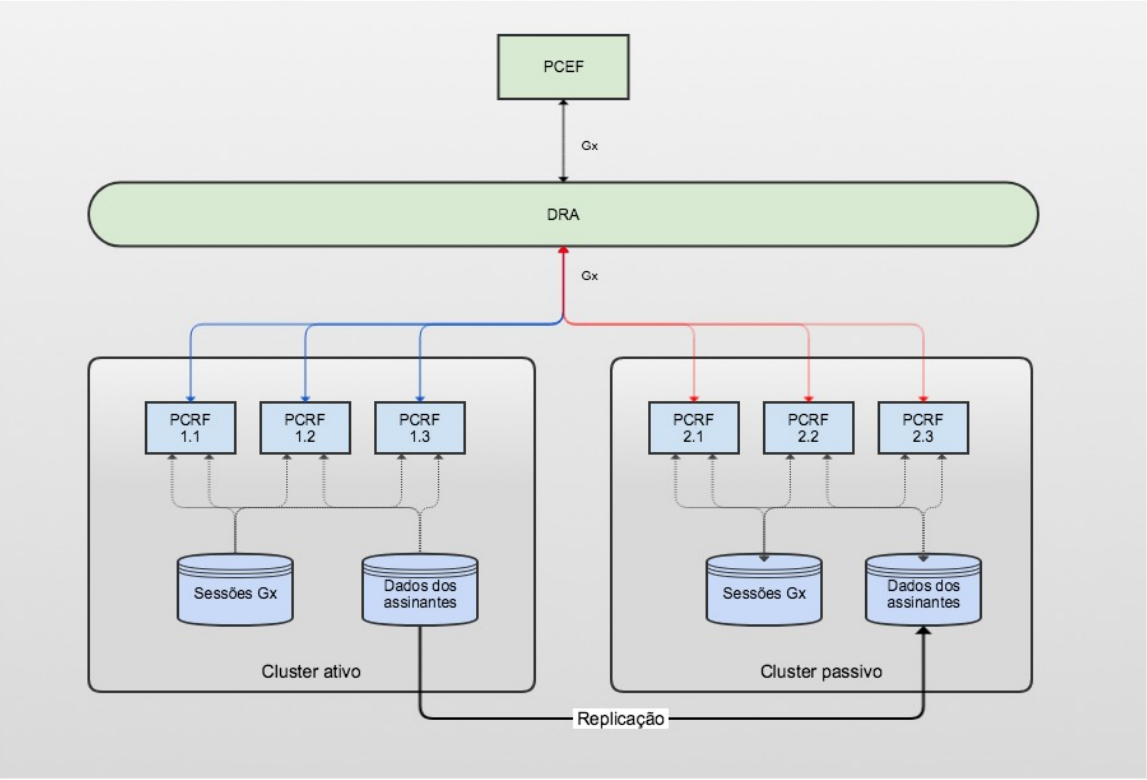
Em uma configuração de *cluster* Ativo-Passivo, a solução executa em um dos sistemas, enquanto o outro sistema fica em modo de espera, pronto para tomar o lugar da solução em caso de falha do primeiro. Quando o *cluster* ativo não é capaz de garantir o

serviço, o segundo sistema assume os recursos apropriados. Este processo é normalmente chamado de *failover*. O segundo sistema substitui totalmente o sistema que falhou, sem que o serviço sofra interrupção.

Apesar da configuração Ativo-Passivo, o *cluster* de base de dados possui replicação ativa de dados entre os componentes SPR. A replicação consiste em duas cópias de um único banco de dados residindo em máquinas diferentes. A replicação ocorre para manter a consistência do perfil do assinante e não haver perdas de controle dos dados de cada assinatura ou atualização do perfil, caso ocorra um *failover*.

A Figura 8 representa o diagrama completo do sistema implementado. Os elementos representados na cor verde são elementos externos ao sistema que foram implementados por outras companhias. O DRA, representado nessa figura como um único bloco, é o ponto comum de comunicação entre o PCRF e o PCEF. Também, apesar dos componentes PCRF, Sessões Gx e Dados serem representados individualmente, eles são diferentes componentes dentro de um mesmo *hardware* como já descrito.

Figura 8: Diagrama completo do sistema

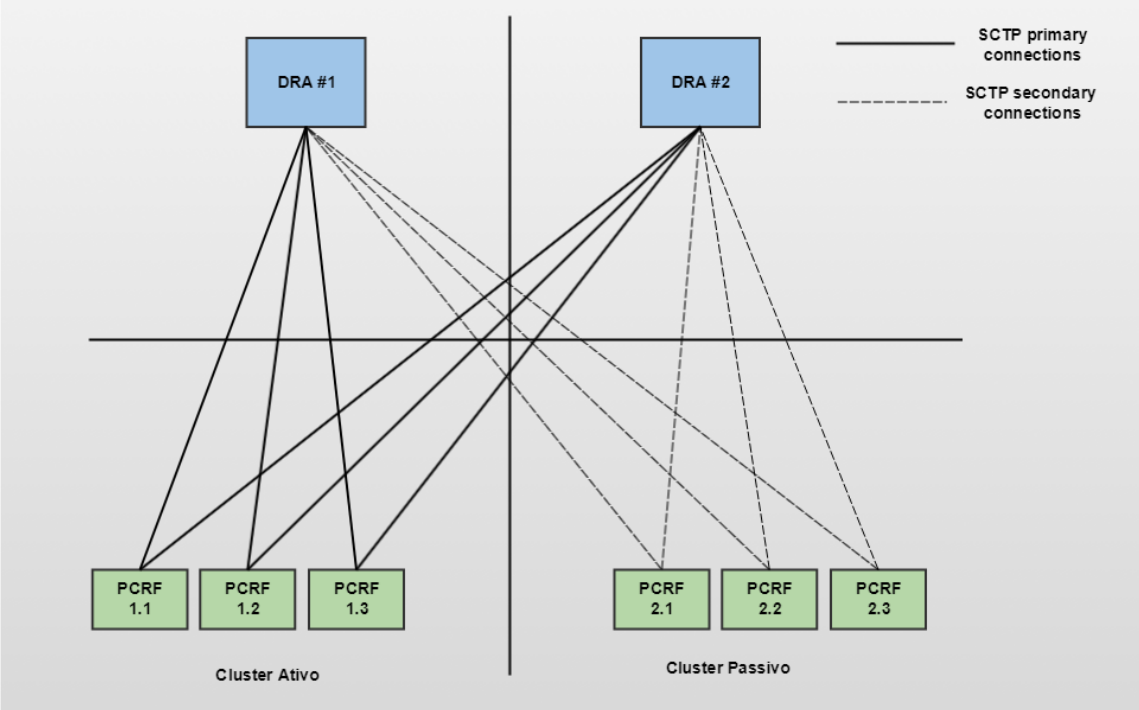


4.2 Integração com o DRA

Como visto na Seção 3.2.1, todos os PCRF do sistema possuem conexões ativas com o DRA, cabendo a este priorizar as conexões com o *cluster* ativo. No sistema anterior da provedora de telefonia móvel, já haviam dois DRA implementados com redundância em

paralelo Ativo-Ativo. Nessa configuração, os dois DRA desempenham os processos juntos, mesmo que apenas uma seja suficiente. Esse caso especial de redundância também poder ser visto como N+1 em paralelo.

Figura 9: Conexões SCTP entre o PCRF e os *peers* do DRA



A Figura 9 detalha as conexões do *cluster* de aplicação com o DRA. Em um cenário que todas as máquinas do *cluster* ativo viessem a falhar, ou fossem derrubadas propositalmente, o DRA automaticamente tornaria prioritárias as conexões do *cluster* passivo e passaria a direcionar o tráfego para este.

4.3 Cenários de falha

Após introduzido o sistema implementado e sua configuração, voltemos agora a atenção para os cenários em que o sistema pode ser considerado em estado de falha, necessitando a comutação para o *cluster* secundário, processo este denominado *failover*.

Cluster de aplicação

Visto que o *cluster* local possui redundância N+1, a perda de um PCRF não caracteriza um cenário de *failover*. Porém, a continuidade do serviço não pode ser garantida em caso de mais de um nó de PCRF estiver *down*. Isso acontece pois apenas um nó de PCRF não seria capaz de lidar com todo o tráfego *Diameter* que seria estabelecido. O sistema até funcionaria, em partes, mas muito dos *requests* provindos do PCEF não seriam atendidos com a prioridade necessária.

Portanto, a perda de mais de um nó de aplicação implica num cenário de *failover*.

Cluster de base de dados

A Tabela 3 representa a distribuição de dados através do *cluster* local. Cópias dos dados são feitas de tal maneira que o sistema possua redundância N+1. Note que ainda que um nó de base de dados falhe, o *cluster* ainda terá uma cópia de todos os dados e não há impacto na continuidade do serviço. A perda de uma base de dados, portanto, não requer migração para o outro site. Porém, é necessária a migração para o sistema redundante caso mais que uma VM de base de dados venha a falhar.

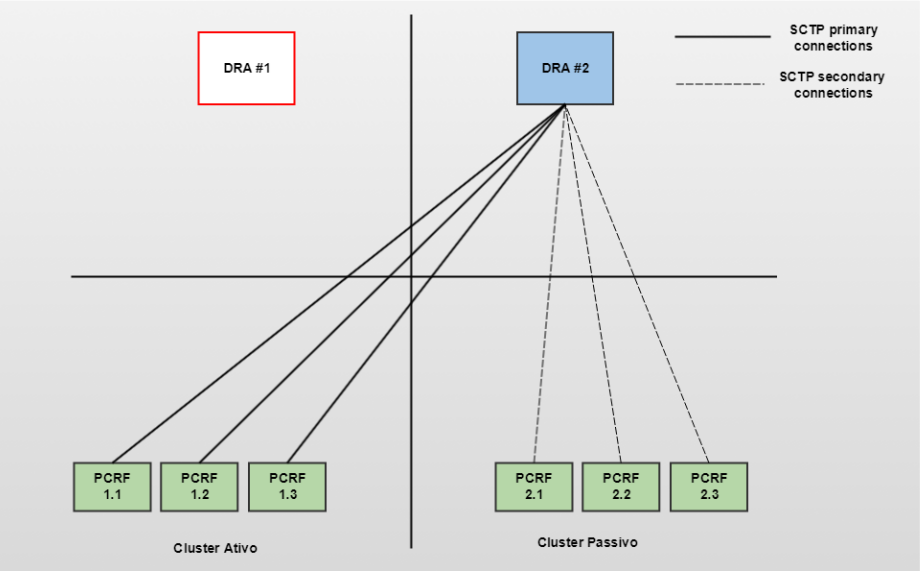
Tabela 3: Distribuição de dados através das runtimes

DB 1		**DB 2**		**DB 3**	
A	C’	B	A’	C	B’
D	F’	E	D’	F	E’
G	I’	H	G’	I	H’
J	L’	K	J’	L	K’

Falhas no DRA

O DRA, como qualquer outro componente, também reside em uma máquina e também é sujeito a falhas. Apesar de ser implementado por outra companhia, uma falha no DRA implicaria diretamente em uma falha do sistema, pois é o DRA que garante a comunicação do sistema com o PCEF.

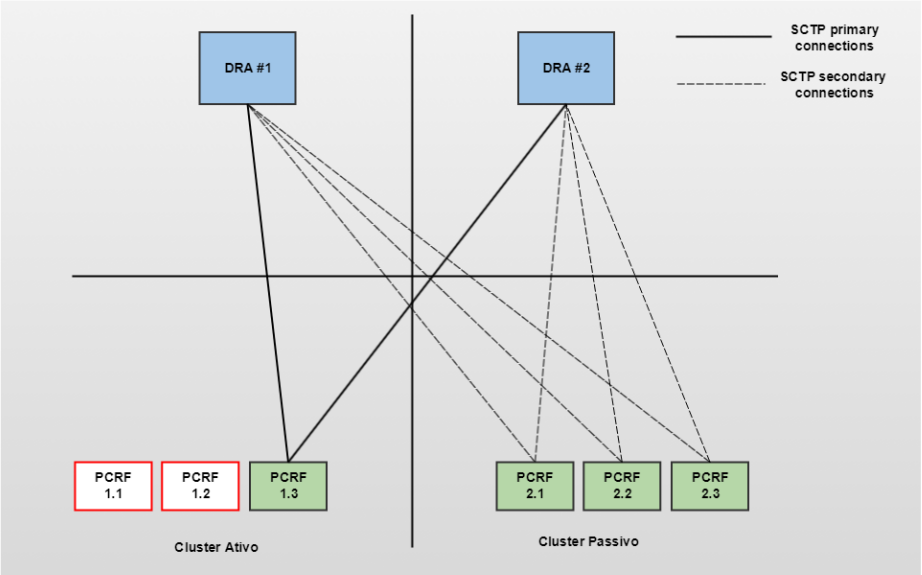
Figura 10: Representação da falha de uma maquina de DRA



Perda de mais de um *peer*

A Figura 11 demonstra o caso em que dois PCRF perderam a conexão com o *peer* do DRA. Isso pode ocorrer devido a falhas no DRA ou nas máquinas de PCRF. Nesse caso, todo o tráfego é direcionado ao PCRF remanescente que não é capaz de lidar com o tráfego sozinho.

Figura 11: Representação das conexões com o DRA quando duas máquinas de PCRF estão *down*



No cenário da Figura 11, um procedimento manual ou automatico deve ser tomado para que o PCRF ativo seja derrubado para o estado *down*. Dessa forma, o DRA passaria imediatamente a transferir o tráfego para o *cluster* passivo. Essa situação é demonstrada na Figura 12.

Perda de uma máquina de DRA

Pode ocorrer a falha de uma das máquinas do DRA, o que implicaria em todo o tráfego do *cluster* primário passar a ser direcionado para o DRA secundário, que é capaz de lidar com o tráfego total graças a redundância do DRA. Esse cenário, representado na Figura 13, portanto, não implica em um cenário de *failover*.

Figura 12: Representação das conexões com o DRA quando são transferidas para o *cluster* secundário

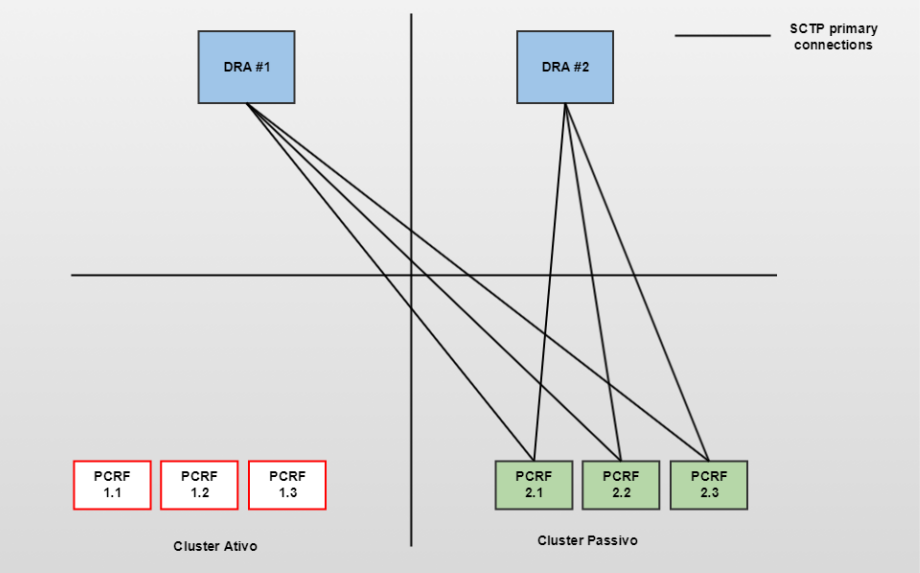
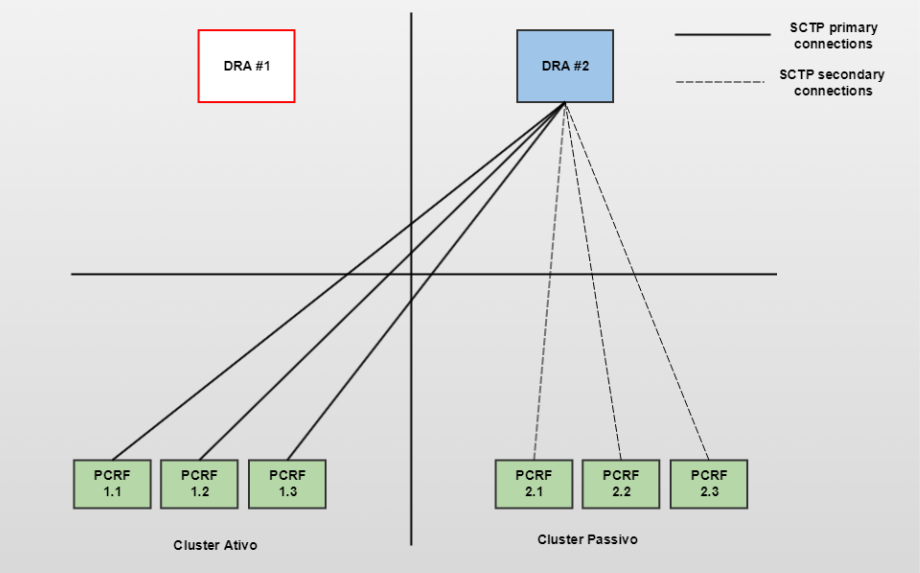


Figura 13: Falha de uma máquina de DRA



5 A FERRAMENTA DE TOLERÂNCIA A FALHAS

Tolerância a falhas é a capacidade de um sistema de recuperar-se de uma falha de um componente sem interrupção do serviço. Um sistema tolerante a falhas possui componentes redundantes que monitoram-se entre si, de forma que a transição para o cluster passivo, diante de uma falha, ocorra quase instantaneamente. Esta definição de tolerância a falhas por Jayaswal (JAYASWAL, 2005) pode ser aplicada para a ferramenta descrita neste capítulo. O *Automatic Failover Tool* (AFT) é a ferramenta de tolerância a falhas desenvolvida nesse trabalho para a empresa de telecomunicações de modo a otimizar a alta disponibilidade do sistema de seu cliente. Utilizando apenas ferramentas *open source*, o AFT é genérico e pode ser utilizado para qualquer tipo de aplicação que envolva monitoramento de componentes através de *requests HTTP* e/ou *SNMP Traps*, conceitos estes que serão definidos na próxima seção.

Ainda segundo Jaywaswal, a primeira regra para garantir alta disponibilidade de uma aplicação é evitar ao máximo intervenção manual. Deve ser possível iniciar, parar e monitorar a aplicação sem qualquer assistência de um operador. Se um aplicativo não pode ser executado em um servidor, ele deve começar rapidamente em outro servidor no cluster. Se iniciar um aplicativo requer uma interface GUI ou executar um script interativo, pode levar horas para alguém para fazer login no sistema e realizar o trabalho necessário. Se o sistema não pode ser acessado remotamente, alguém deve fisicamente chegar ao escritório ou *data center*. O *hardware* também pode estar localizado em um lugar geograficamente isolado, caso em que pode ser difícil para encontrar alguém com experiência requerida rapidamente.

Por último, Moser (MOSER, 2004) define *failover* como o processo no qual uma máquina assume os serviços de outra, quando esta apresenta falha. O *failover* pode ser automático ou manual, sendo o automático o que normalmente é esperado de uma solução de alta disponibilidade. Porém, algumas soluções não críticas podem tolerar um tempo maior até que o serviço se recupere, podendo utilizar técnicas de *failover* manual.

5.1 Definições

5.1.1 Web service

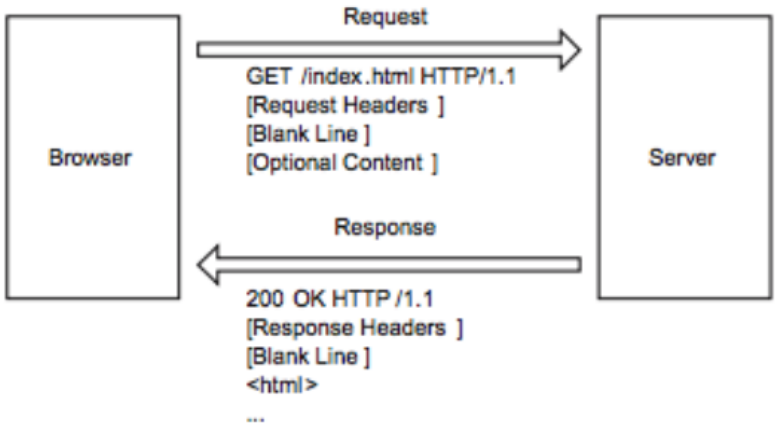
A W3C, principal organização internacional de padronização da *World Wide Web* (WWW), define *Web service* como um *sistema de software* projetado para suportar a interação entre máquinas através da rede (W3C, 2004). Diferente de *software*, que normalmente se refere a um conjunto de instruções que executam uma tarefa específica, um *sistema de software* se refere a um conceito mais abrangente, envolvendo outros

componentes, como especificações, resultados de testes, documentação para o usuário final, dentre outros (GRUB; TAKANG, 2003).

Em outras palavras, *Web service* é um serviço oferecido por uma máquina para outra máquina, possibilitando a comunicação entre elas através da *Web*. Em um *Web service*, um protocolo de comunicação voltado para a *Web*, como o HTTP, é utilizado para a comunicação entre máquinas, mais especificamente para transferir arquivos de leitura entre elas. Segundo Gurugé (2004), *Web services* operam através da troca de dados no formato XML. Em verdade, *Web services* do tipo *SOAP* (*(Simple Object Access Protocol*,em português Protocolo Simples de Acesso a Objetos) trocam dados apenas no formato XML. Os do tipo *REST* podem também trocar dados no formato JSON, dentre outros.

O HTTP é outro protocolo de comunicação pertencente a camada de aplicação, assim como o *Diameter*. Ele funciona como um protocolo de *request-response* ou requisição-resposta no modelo computacional cliente-servidor. De acordo com Downey (2007), sempre que alguém acessa uma página na internet, existe comunicação entre dois computadores. Em um deles há um *software* conhecido como navegador e no outro, outro *software* conhecido como *web server*. O navegador envia um *request* para o servidor e o servidor envia a resposta para o navegador. A requisição contém o nome da página que está sendo requisitada e a resposta contém a própria página em si que está sendo acessada (caso esteja disponível). Esta situação é representada na Figura 14.

Figura 14: *Request/response* trocados entre um navegador de internet e um *website* (server)



Fonte: Downey (2007)

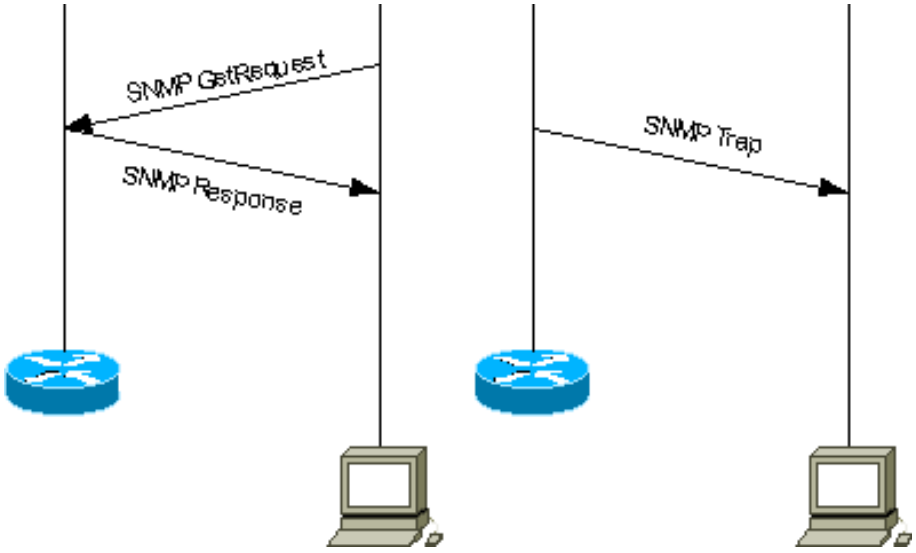
5.1.2 SNMP traps

O *Simple Network Management Protocol* (SNMP) é um protocolo de rede padrão, pertencente a camada de aplicação e integrante do conjunto de protocolos TCP/IP, e tem como finalidade o gerenciamento de dispositivos em uma rede IP. As principais

funcionalidades do protocolo consistem em monitoramento dos dispositivos da rede e configuração remota de parâmetros nos mesmos, tanto de forma manual ou quanto automaticamente em resposta a um incidente determinado (ALVARENGA; RAMOS, 2011).

Dispositivos gerenciados podem enviar notificações SNMP para seus gerentes quando certos eventos ocorrem. Um exemplo de notificação importante que um SNM (*System and Network Monitor*) pode receber seria de uma falha em algum dos roteadores. *Trap* é um dos dois tipos de notificação que o SNMP suporta. O envio de uma mensagem *Trap* permite que um agente notifique o sistema de gerenciamento para a ocorrência de qualquer evento relevante em qualquer instante de tempo. No entanto, não há qualquer confirmação por parte do gerente do recebimento da *Trap*. De acordo com Mauro e Schmidt (2001), *Traps* são um método para que um agente envie a uma estação de monitoração uma notificação assíncrona sobre as condições que o monitor deve conhecer. Em mensagens assíncronas, o remetente não espera por uma resposta.

Figura 15: Tipo de mensagens trocadas entre cliente e servidor SNMP



Fonte: Cisco (2006)

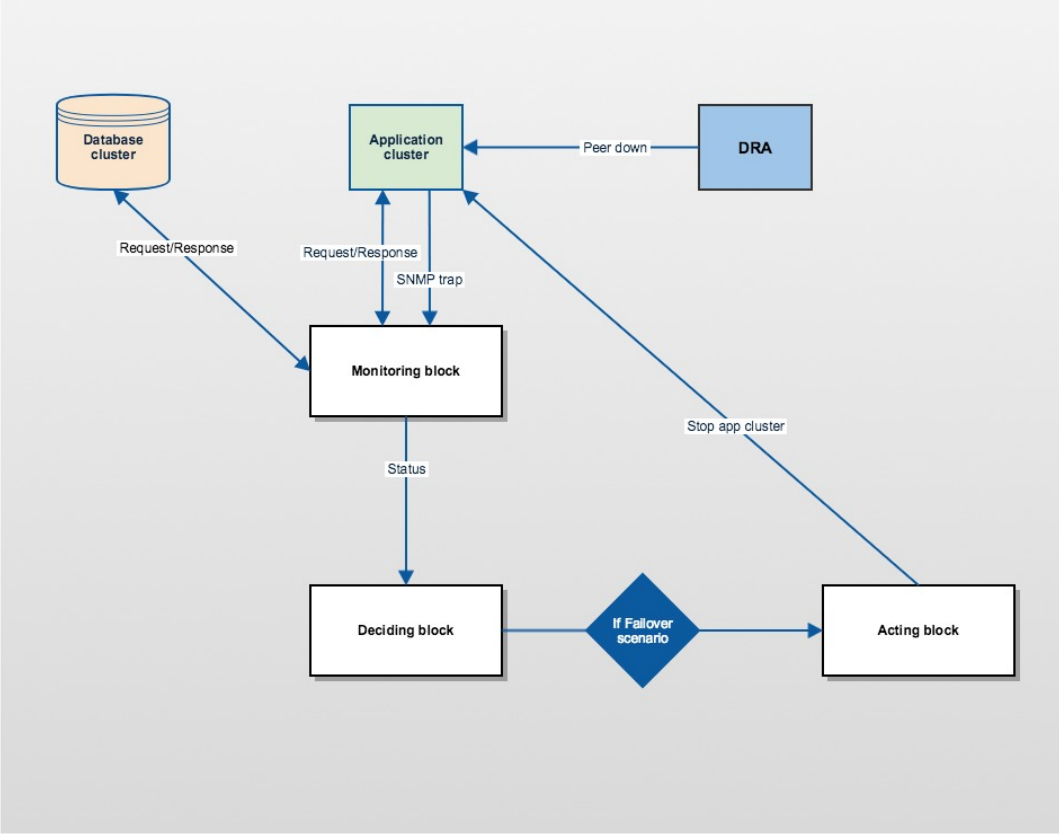
No AFT, *Traps* são utilizadas para monitorar as conexões entre as máquinas de PCRF e as máquinas de DRA. A solução é capaz de gerar *traps* em diversas situações de erro, como a queda de um *peer*, e enviá-las a servidores configurados na solução.

5.2 Automatic Failover Tool

5.2.1 Design

Foi requisitado pelo cliente uma ferramenta que rapidamente detectasse situações de falha no *cluster* ativo e caso os critérios fossem atingidos, iniciasse automaticamente o processo de *failover* para o *cluster* passivo.

Figura 16: Diagrama de alto nível de projeto do AFT

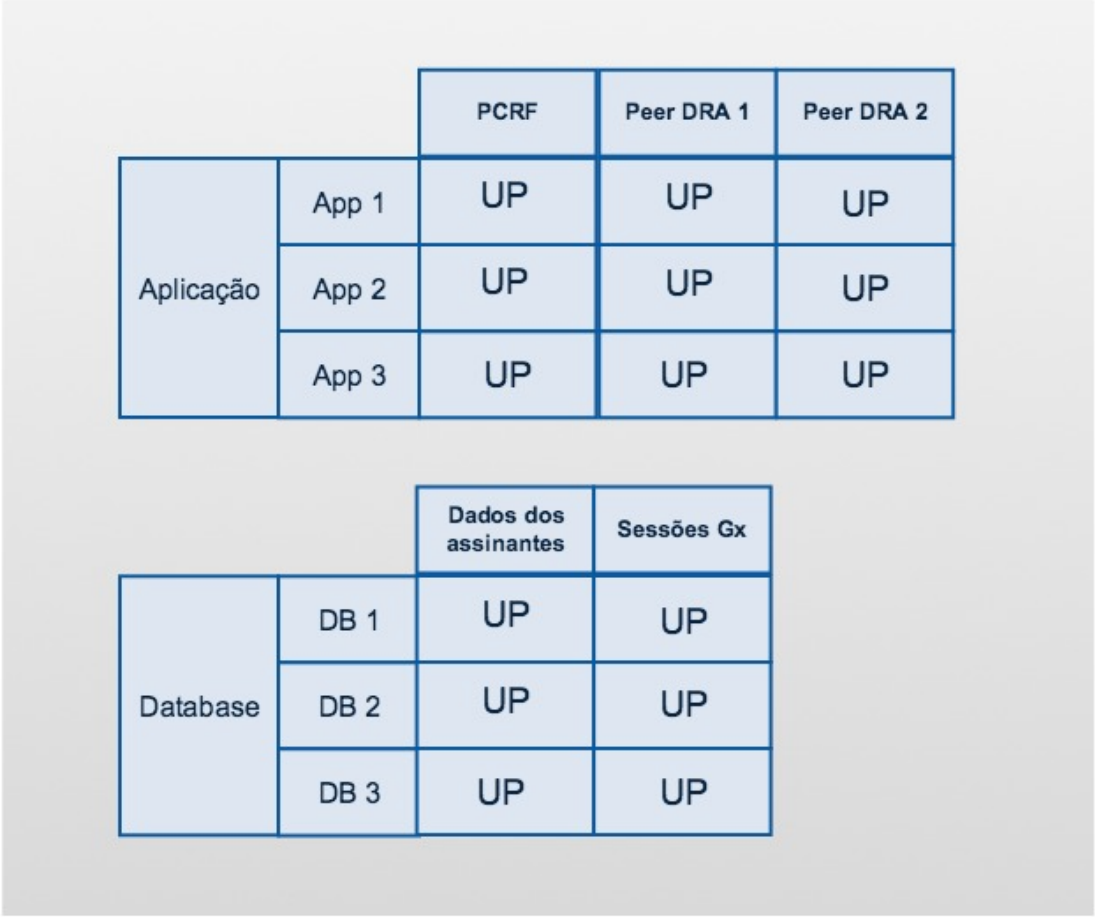


A ferramenta foi projetada inicialmente com três principais blocos, representados na Figura 16 Um bloco de monitoramento (Monitoring) que seria responsável por monitorar o estado dos componentes rodando nas máquinas do *cluster* ativo. Foi definido que seriam utilizados *requests* HTTP para a monitoramento do estado desse componentes. Um servidor HTTP, que não faz parte do escopo desse trabalho, foi instalado nas máquinas e configurado para que ao receber *requests* HTTP do tipo GET, respondesse com o estado *up* ou *down* dos componentes rodando naquela máquina. O bloco de monitoramento também seria o responsável por monitorar as conexões SCTP com os *peers* do DRA. A solução é capaz de gerar SNMP *traps* em diversas situações de erro e enviá-las para servidores pré-configurados.

Definiu-se que haveria uma estrutura de dados principal encapsulada em uma das classes que conteria o estado de todos os elementos monitorados. Essa estrutura de dados, denominada sumário e representada na Figura 17, seria acessada então pelo bloco de decisão (Deciding) que seria capaz de julgar se o cenário atual é ou não um dos cenários de *failover*.

Em caso positivo, o bloco de decisão acionaria o bloco de ação (Acting), que tomaria uma ação de iniciar a migração para o *cluster* passivo. Como visto na Seção 4.2, basta que as máquinas do *cluster* ativo sejam derrubadas para que o DRA automaticamente

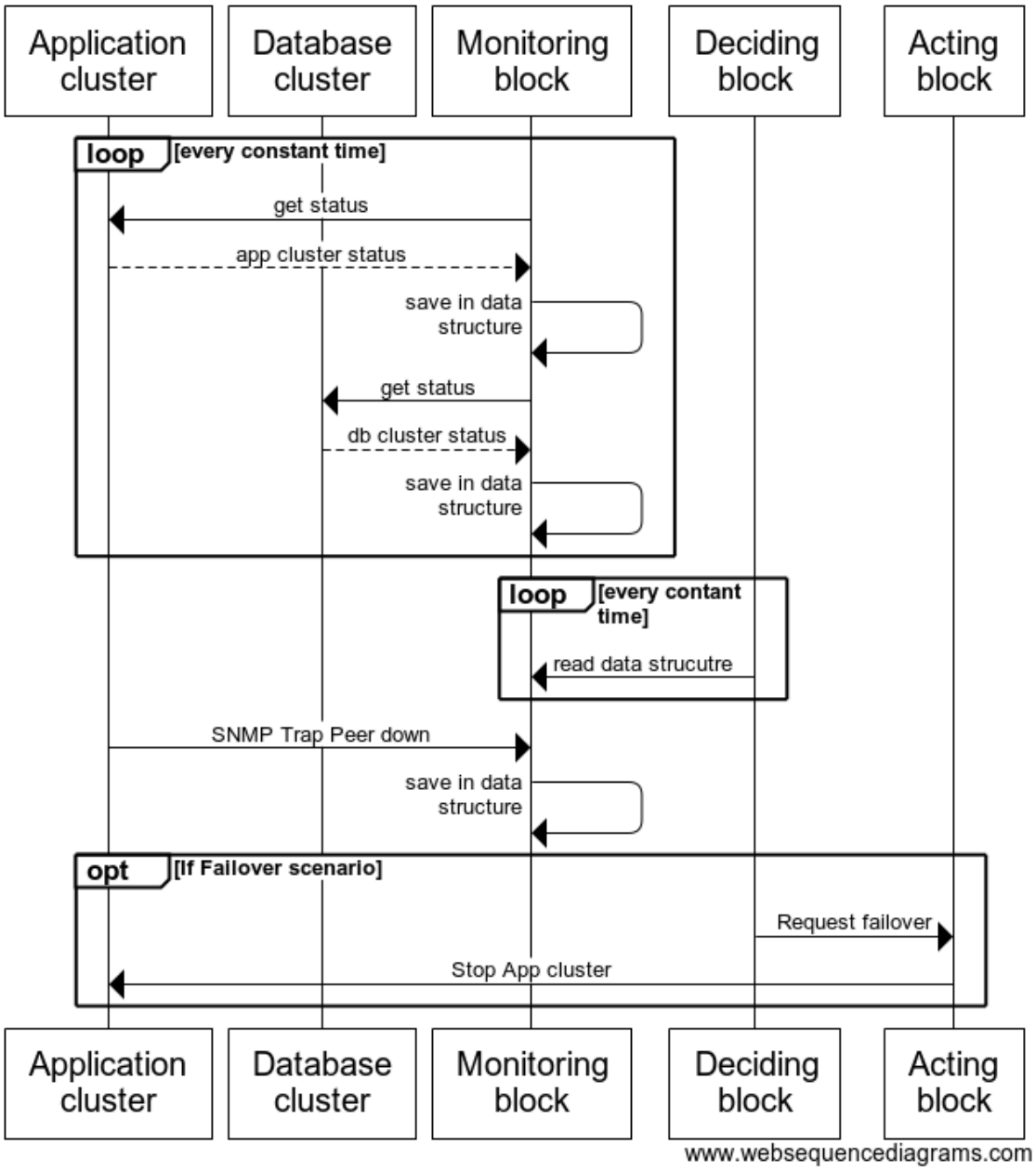
Figura 17: O sumário, a principal estrutura de dados



comece a redirecionar o tráfego para o *cluster* secundário. Determinou-se que o bloco de ação agiria exatamente dessa maneira, disparando um *request* HTTP nas máquinas de aplicação e derrubando todos os componentes. Não haveria a necessidade de realizar nenhum comando no *cluster* de base de dados, pois essas máquinas no *cluster* secundário já estão ativas devido a replicação de dados.

O diagrama de sequência representado na Figura 18 demonstra os principais processos que ocorrem na ferramenta. A fim de agilizar o processo de detecção, *threads* distintas seriam utilizadas para as principais classes. *Threads* são tarefas independentes dentro de um mesmo processo e executadas em paralelo. Como todas as classes direta ou indiretamente acessariam o sumário, foi implementado o conceito de *locking* nas *threads*, ou seja, apenas uma *thread* por vez pode interagir com a estrutura, evitando assim conflitos, como por exemplo uma *thread* executar a leitura da estrutura enquanto outra *thread* distinta está atualizando a mesma estrutura.

Figura 18: Diagrama de alto nível de projeto do AFT



5.2.2 Classes

Esta seção descreve as classes que foram implementadas de modo a atingir os objetivos descritos no *design*.

Classe Requests

A classe Requests possui métodos que enviam *requests* do tipo GET e PUT para o *web server* que foi implementado nas máquinas do *cluster* ativo, interpretam a resposta e podem utilizar essa informação para atualizar o sumário.

Os métodos da classe Request são representados na Figura 19 pelos blocos internos

ao bloco Request. A Figura 20 demonstra o diagrama de sequência para essa classe.

Os métodos “GET status DB” e “GET status App” enviam *requests* do tipo GET aos *web servers*. É pré-configurado no *server* que a resposta pros *requests* GET seja o estado *up* ou *down* dos componentes que rodam na máquina. Quando o *request* GET chega no servidor, a máquina executa comandos que verificam se o componente está funcionalmente rodando ou não. O resultado desses comandos é então transformado em respostas para o *request* e devolvida ao cliente.

Podem haver ocasiões em que a máquina não seja capaz de executar esses comandos e responder o *request* apropriadamente. O *request* pode então falhar e nesse caso as máquinas seriam consideradas *down* pelos métodos de análise da resposta.

Além do estado de falha, os *requests* também podem apresentar o estado de *timeout*. *Timeout* é um parâmetro de rede relacionado a eventos que foram projetados para ocorrer até a conclusão de um tempo decorrido pré-determinado. No contexto do AFT, *timeout* é o tempo em que o AFT aguarda até uma máquina responder o seu *request*. O *timeout* no AFT é parametrizável no código através da constante TIMEOUT e foi definido em cinco segundos. Caso o servidor não tenha respondido o *request* nesse período, o AFT tentará executar o *request* outras vezes antes de considerar os componentes dessa máquina com o estado *down*. Essa quantidade de vezes também é parametrizável pela constante TIMEOUT_MAX_RETRY, que tem valor igual a três. Outra constante de tempo importante no AFT, denominada TIME_BETWEEN_REQUESTS, é o tempo em que a classe Requests leva para executar novamente os *requests* nas máquinas. As constantes de tempo são representadas na Tabela 4.

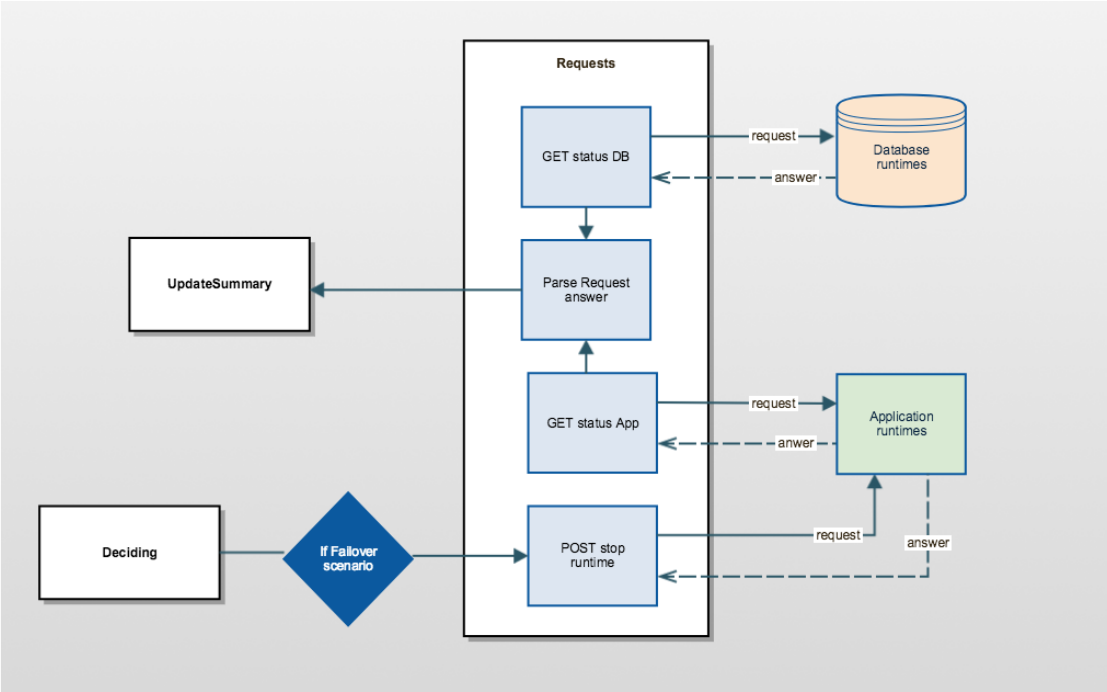
Tabela 4: Constantes de tempo da ferramenta

Constante de tempo	**Valor (s)**
TIME_BETWEEN_REQUESTS	5
TIME_BETWEEN_DECISIONS	30
REQUEST_TIMEOUT	5
TIME_PEERS	10

Classe TrapReceiver

A classe TrapReceiver, representada na Figura 21, possui dois métodos: SNMP Trap Server e Parse Trap. O método SNMP Trap Server é basicamente um SNMP Notification Receiver. É um servidor que recebe *traps* que a solução é capaz de gerar em situações de erro. Diversos tipos de *traps* da solução podem chegar nesse servidor, mas apenas *traps* referentes aos estados dos *peers* que conectam as máquinas de aplicação com as máquinas de DRA são interessantes para o AFT. É por isso que toda *trap* que chega nesse servidor é enviada ao método Parse Trap, que analisa e identifica a *trap*. Caso esta seja o estado de

Figura 19: Classe Request e seus principais métodos



algum dos *peers*, o método envia a informação para a classe UpdateSummary, que atualiza o sumário. A Figura 22 contém o diagrama de sequência com os processos descritos.

Classe UpdateSummary

A classe UpdateSummary é onde está encapsulada a variável que armazena a principal estrutura de dados do AFT, o sumário. Este, já representado na Figura 17, contém o estado *up* ou *down* de todos os componentes envolvidos no processo de monitoramento. A classe, representada na Figura 23, possui métodos que, ao serem invocados por outras classes, atualizam os estados dos componentes no sumário.

Um arquivo YAML de *input* é utilizado pelo método "Read Initial Summary"para carregar o sumário com valores iniciais e a cada mudança registrada, o método "Dump Summary"escreve outro arquivo YAML de *output*. A utilidade desse arquivo é no caso de o AFT precisar ser interrompido por qualquer motivo e depois reiniciado com os últimos valores registrados do sumário.

O conceito de *locking* é muito importante nessa classe, pois como representado na Figura 24, diferentes processos interagem com o sumário ao mesmo tempo. O sumário é atualizado pelos métodos "App Update Status", "Db Update Status"e "Peer Update Status"com os estados das máquinas de aplicação, base de dados e das conexões com os *peers* do DRA.

Figura 20: Diagrama de sequência da classe Requests

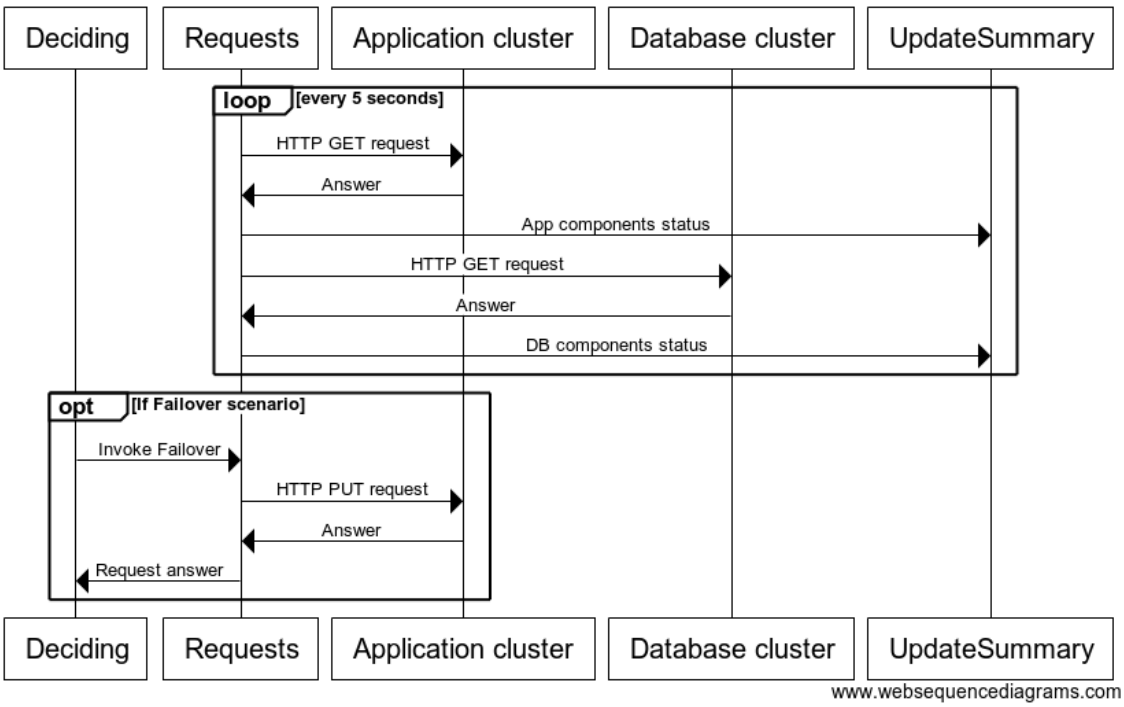
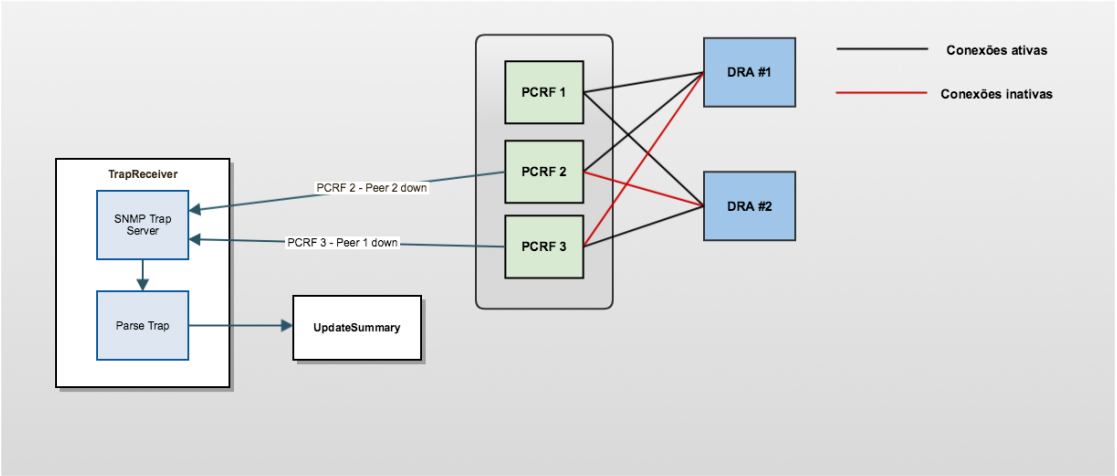


Figura 21: Classe TrapReceiver e seus principais métodos

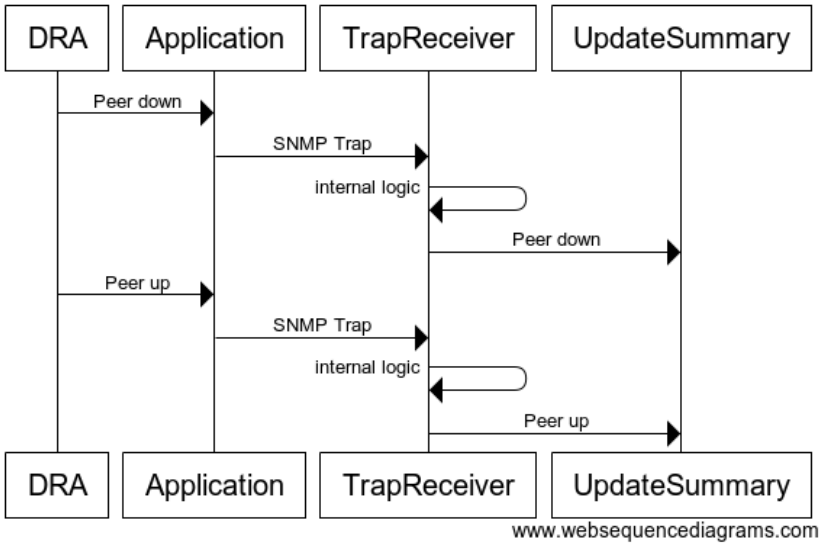


Classe Deciding

A classe Deciding é a responsável por interpretar o sumário e julgar se o cenário atual representa um cenário de *failover* ou não. Ela possui o método "decide_current_scenario" que executa um *loop* em uma *thread* individual a cada 30 segundos. Esse valor é parametrizável e representa outra constante de tempo do AFT chamada de TIME_BETWEEN_DECISIONS (Tabela 4).

O método lê as colunas do sumário e procura pelos cenários de *failover* descritos na Seção 4.3. Os cenários são relativamente simples com exceção do cenário representado pela Figura 13, em que ocorre a falha de uma das máquinas de DRA, o que não caracteriza

Figura 22: Diagrama de sequência da classe TrapReceiver



um cenário de *failover*. Esse cenário pode ser facilmente confundido com o cenário da Figura 11, em que mais de uma máquina de PCRF perdeu a conexão com o *peer* do DRA e é considerado um cenário de *failover*. Por isso, foi introduzida a constante de tempo parametrizável TIME_PEERS em que o bloco, ao identificar um *peer down*, aguarda por mais 10 segundos até uma atualização do sumário sobre os outros *peers*, para ter certeza se o que caiu foi de fato um ou mais *peers* ou uma das máquinas de DRA.

Tabela 5: Cenários de *failover* com base nos nós de aplicação e base de dados

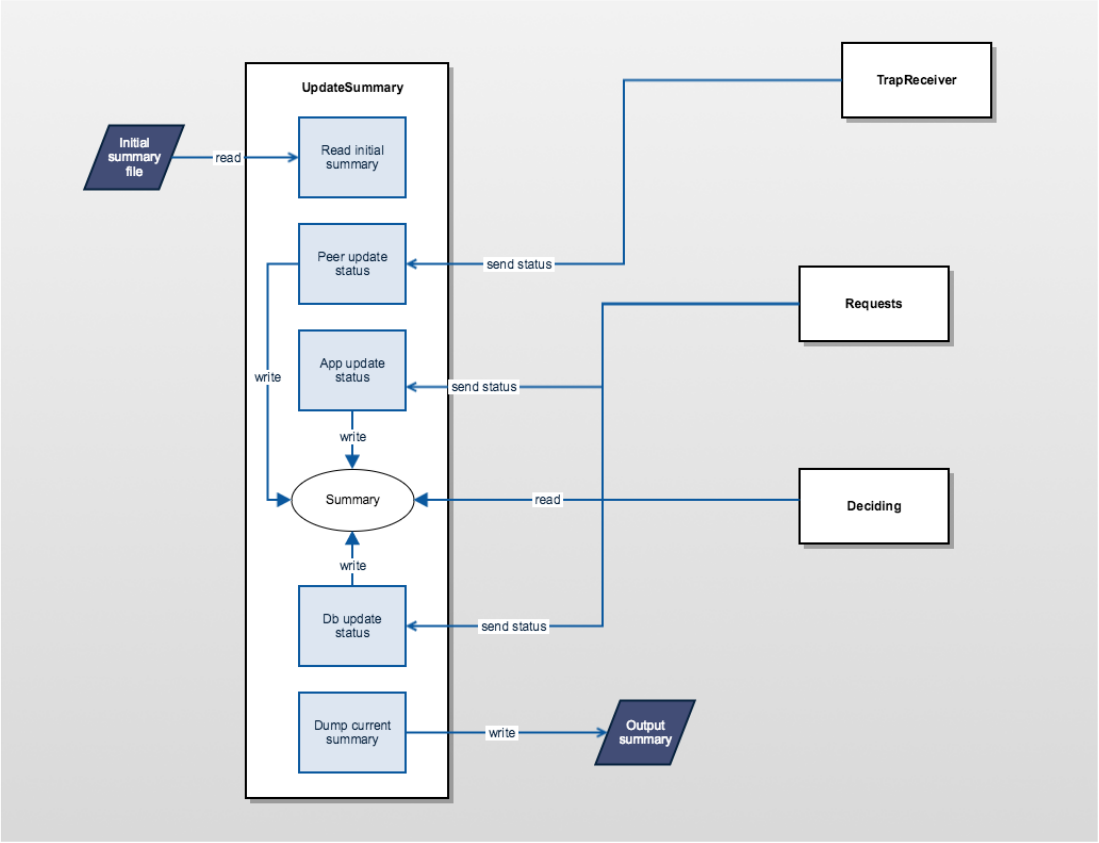
Cenário	**Cenário de *failover***
Perda de um nó de aplicação	Não
Perda de mais de um nó de aplicação	Sim
Perda de um nó de base de dados	Não
Perda de mais de um nó de base de dados	Não

Tabela 6: Cenários de *failover* com base nos componentes

Cenário	**Cenário de *failover***
Queda de um componente PCRF	Não
Queda de mais de um componente PCRF	Sim
Queda de um componente Sessões	Não
Queda de mais de um componente Sessões	Sim
Queda de um componente Dados	Não
Queda de mais de um componente Dados	Sim

Quando uma situação de *failover* é detectada, o bloco Deciding aciona um método da classe Requests que imediatamente faz um *request* do tipo PUT nas máquinas de aplicação do *cluster* ativo. O request executa um comando nas máquinas que derruba todos os componentes. Quando o DRA nota que perdeu todas as conexões com o *cluster* primário, ele automaticamente passa a direcionar o tráfego para o *cluster* secundário (12).

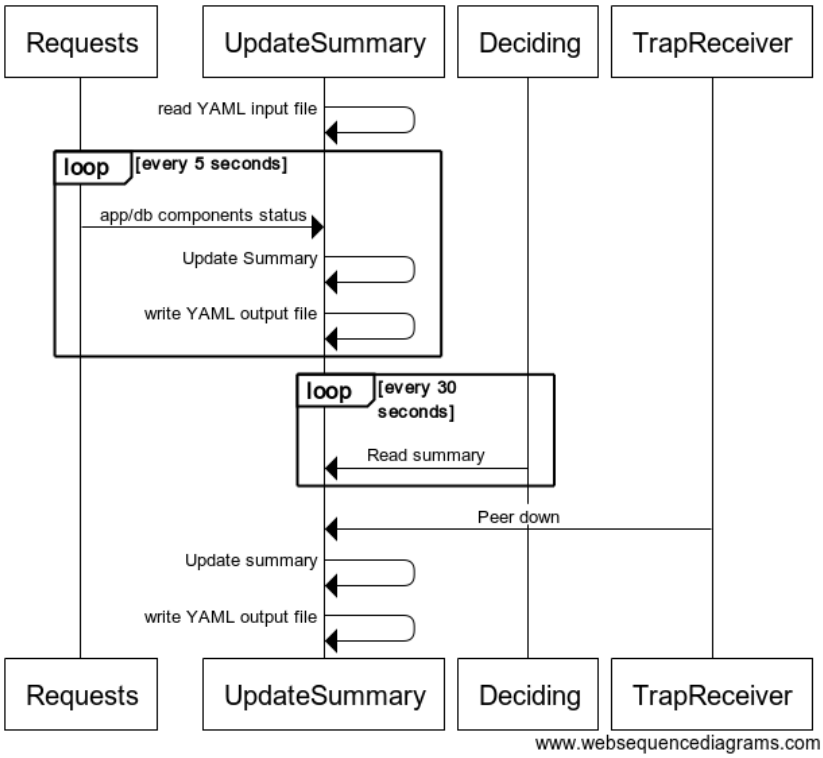
Figura 23: Classe UpdateSummary e seus principais métodos



Nás máquinas de base de dados, nenhuma operação necessita ser realizada, devido a conexão ativo-ativo entre os *clusters*.

Procedimentos manuais devem ser tomados para reparar as falhas no cluster que falhou, não sendo parte do escopo do AFT reparar ou evitar nenhum tipo de erro.

Figura 24: Diagrama de sequência da classe UpdateSummary



6 RESULTADOS

Durante o tempo de implementação, foi possível testar o AFT no ambiente de produção em que ele seria de fato aplicado. Os testes realizados na ferramenta resumiam-se em "derrubar" propositalmente os componentes nas máquinas em que o AFT monitorava. O objetivo dos testes não era somente verificar se o AFT reconheceria o cenário de *failover*, como também quantificar o tempo aproximado em que esse processo ocorria. A Tabela 7 contém o tempo registrados nos *logs* de alguns eventos. Além disso, a Figura 25 demonstra as quatro principais classes do AFT descritas no Capítulo 5 e os principais processos que ocorrem no AFT.

Figura 25: Diagrama de alto nível de projeto do AFT

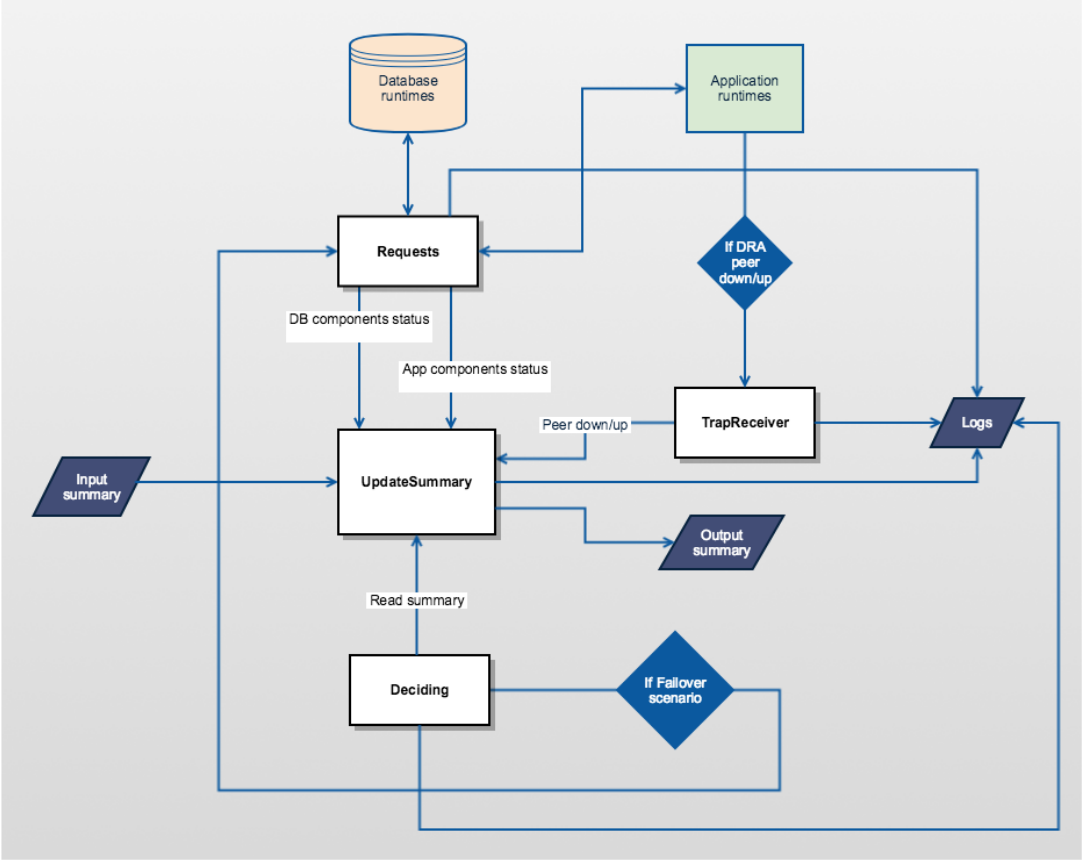


Tabela 7: Tempo registrado entre eventos de falha e a detecção dos mesmos

Evento	**Tempo (ms)**
Detecção da queda de um componente de aplicação	11
Detecção da queda de um componente de database	20
Tempo até o *failover*	17,78

Como visto no Capítulo 2, a Equação 2.1 calcula a disponibilidade de um sistema

utilizando dois parâmetros, o MTBF e o MTTR. Após apresentada a ferramenta de *failover* automático, podemos concluir que o AFT está diretamente relacionado ao parâmetro MTTR, ou *Mean Time to Repair*. Isto porque o AFT não é uma ferramenta que possui a função de evitar falhas, ou minimizá-las de qualquer forma e sim, uma ferramenta que possui uma resposta a falhas. Vale evidenciar que o AFT não repara nenhum tipo de falha nos componentes ou na máquina em que ela ocorreu, mas ao realizar a migração para o *cluster* secundário, o AFT repara o sistema como um todo, minimizando o seu tempo *down*.

Um procedimento manual se torna necessário para realizar a migração para o *cluster* secundário sem a utilização de uma ferramenta automática. Além disso, o processo de detecção do cenário de falha é consideravelmente demorado sem o uso de uma ferramenta de detecção. É conhecido que a solução da companhia envia registros de *status* a cada cinco minutos para um servidor do cliente. No pior caso, a falha ocorreria no momento em que um registro foi enviado e levaria outros cinco minutos até que fosse enviado outro registro com essa falha. A esse tempo, soma-se o tempo até alguém verificar esse *log* e executar o procedimento manual de *failover*. Sabe-se através do histórico da companhia que esse processo já chegou a levar até 30 minutos.

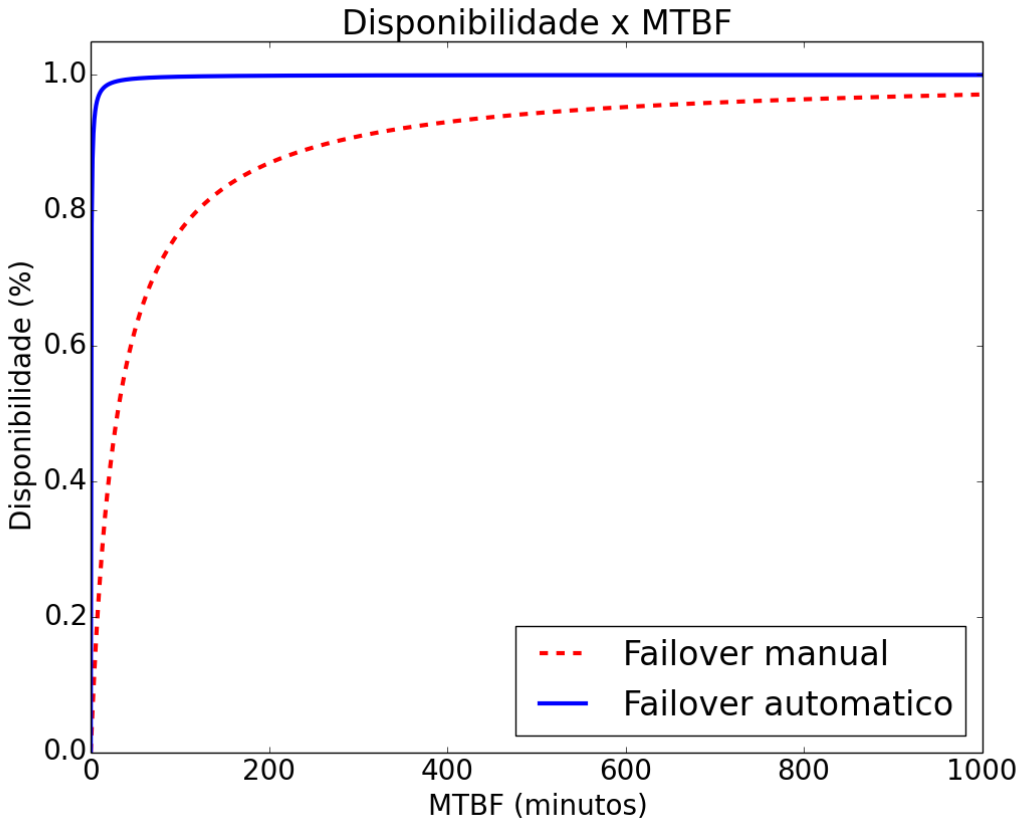
Por outro lado, o AFT pode levar, no pior cenário, cerca de 47 segundos para executar o processo de *failover*. No pior cenário, uma falha ocorre exatamente no momento em que o AFT acabou de tomar uma decisão, e levará agora mais 30 segundos até a próxima. A esse tempo é somado os 17 segundos que o AFT leva para desligar os componentes das máquinas do *cluster* principal. No melhor caso, em contrapartida, o AFT registra a falha no momento em que uma decisão vai ter início e leva apenas 17 segundos para o processo de *failover*.

Com base nesses valores de MTTR dos processos automático e manual, plotou-se a curva de disponibilidade do sistema em função do tempo médio entre falhas (MTBF) para os dois cenários: utilizando o AFT e utilizando o procedimento manual. As curvas estão representadas na Figura 26. Nota-se que para valores baixos de tempo médio entre falhas, ou seja, quando o sistema falha bastante, o uso do AFT aumenta consideravelmente a disponibilidade do sistema em relação ao procedimento manual. Porém, para sistemas que falham muito pouco, o AFT faz pouca diferença na disponibilidade em relação ao procedimento manual.

Também, plotou-se o número de noves em função do tempo médio entre falhas para os dois cenários. O resultado está representado na Figura 27. Nota-se uma grande diferença no número de noves que o AFT garante a solução em relação ao procedimento manual.

Visto que sistemas de telecomunicação normalmente possuem cinco noves de disponibilidade (Tabela 1), o que equivale a um tempo *down* de 5,25 minutos por ano.

Figura 26: Curvas de disponibilidade em função do tempo médio entre falhas utilizando *failover* manual e automático (AFT)



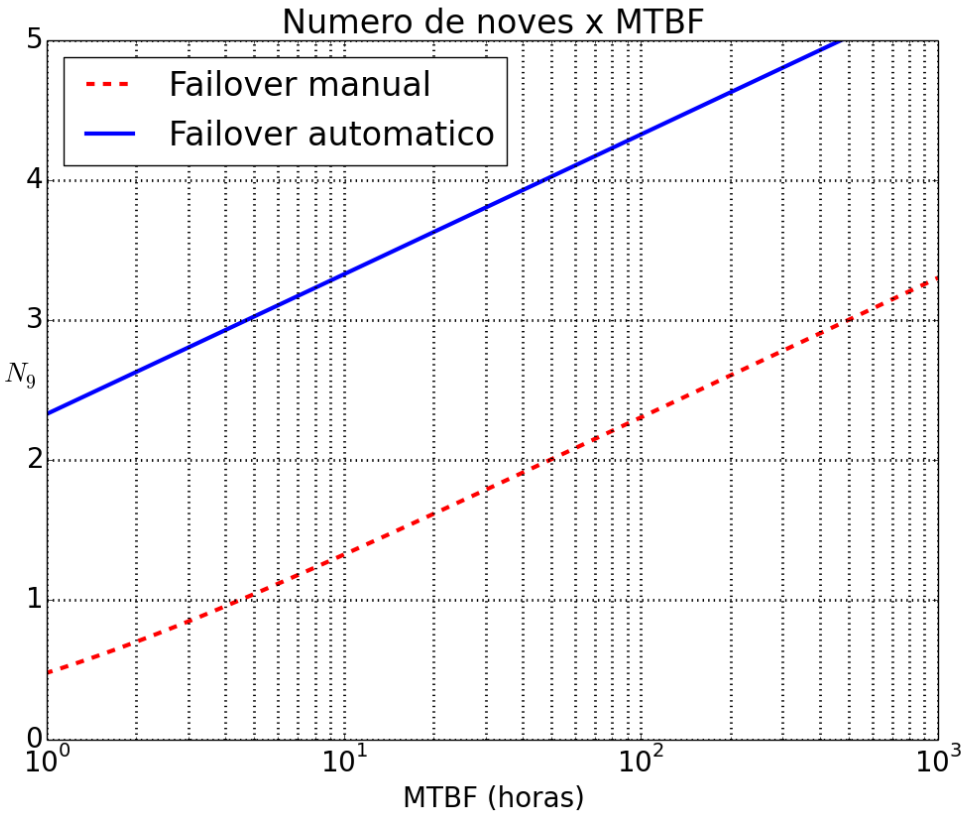
Podemos aplicar uma análise teórica em que o AFT seria capaz de reduzir o tempo de indisponibilidade para o tempo que o AFT leva até o *failover*. Utilizando o melhor e pior caso descritos nos parágrafos anteriores, criou-se a Tabela 8, que representa a disponibilidade atingida pela ferramenta utilizando esses tempos.

Tabela 8: Tempo até o *failover* e disponibilidade atingida no melhor e pior caso

	Tempo até o failover (s)	**Disponibilidade atingida**
Pior caso	47	99,999%
Melhor caso	17	99,9999%

Portanto, é possível visualizar que a ferramenta de tolerância a falhas implementada é capaz de garantir a adição de outro nove na disponibilidade de sistemas de telecomunicação, aumentando ainda mais a disponibilidade de sistemas que já são considerados muito disponíveis.

Figura 27: Número de noves em função do tempo médio entre falhas utilizando *failover* manual e automático (AFT)



7 CONCLUSÃO

A implementação da ferramenta apresentada no trabalho mostrou-se uma solução simples, viável e de baixo custo de alta disponibilidade. Quando aplicada a solução em um ambiente real, foi possível manter a disponibilidade do sistema mesmo diante de cenários de falha.

O tempo de resposta em situações simuladas de falha que ativaram o *failover* foi bastante adequado, ficando abaixo do tempo de um minuto exigido pela companhia.

Outros marcos notáveis alcançados com esta solução foram o emprego de ferramentas *open-source* com todos seus benefícios agregados como menor custo de implementação, menores taxas de manutenção, facilidade de escalabilidade sem aquisição de licenças especiais e uso de tecnologia inovadora.

Também foi notável a exigência do avanço no conhecimento da linguagem de programação e nos conceitos envolvidos.

REFERÊNCIAS

3GPP-23.203. *Policy and charging control architecture*. 2017. Disponível em: <http://www.3gpp.org/DynaReport/23203.htm>. Acesso em: 12.10.2017.

3GPP-29.213. *Policy and charging control signalling flows and Quality of Service (QoS) parameter mapping*. 2017. Disponível em: <http://www.3gpp.org/DynaReport/29213.htm>. Acesso em: 12.10.2017.

3GPP-29.214. *Policy and charging control over Rx reference point*. 2017. Disponível em: <http://www.3gpp.org/DynaReport/29214.htm>. Acesso em: 12.10.2017.

ALLEN, M. **Redundancy: N+1, N+2 vs. 2N vs. 2N+1**. 2017. Disponível em: <http://www.linkedin.com/pulse/20141111155245-3267680-redundancy-n-1-n-2-vs-2n-vs-2n-1/>. Acesso em: 28.9.2017.

ALVARENGA, I. D.; RAMOS, B. L. **Simple Network Management Protocol (SNMP)**. 2011. Disponível em: <http://www.gta.ufrj.br/grad/11_1/snmp/index.html>. Acesso em: 30.8.2017.

AVIZIENIS, A. **Design of fault-tolerant computers**. University of California. Fall Joint Computer Conference: [s.n.], 1998.

BARTON, B. **Gx interface - sitting between PCRF and PCEF**. 2012. LTE AND BEYOND. Disponível em: <http://www.lteandbeyond.com/2012/01/gx-interface-sitting-between-pcrf-and.html>. Acesso em: 10.8.2017.

BATISTA, A. C. Estudo teórico sobre cluster linux. **Universidade Federal de Lavras**, 2007.

CISCO. **Understanding Simple Network Management Protocol (SNMP) Traps**. 2006. Disponível em: <http://www.cisco.com/c/en/us/support/docs/ip/simple-network-management-protocol-snmp/7244-snmp-trap.html>. Acesso em: 30.8.2017.

CNT. Achieving high availability objectives. 2003.

COSTA, H. L. A. **Alta disponibilidade e balanceamento de carga para melhoria de sistemas computacionais críticos usando software livre: Um estudo de caso**. 2009. 95 f. Dissertação (Pós-Graduação em Ciências da Computação) — Universidade Federal de Viçosa, Viçosa, 2009.

DIALOGIC. **Packet Data Network (PDN)**. 2017. Disponível em: <http://www.dialogic.com/glossary/packet-data-network-pdn>. Acesso em: 12.8.2017.

DOWNEY, T. **Web Development with Java**. [S.l.]: Springer, London, 2007.

F5. **Diameter Routing Agent (DRA)**. 2017. Disponível em: <http://f5.com/glossary/diameter-routing-agent-dra>. Acesso em: 10.8.2017.

FILHO, N. A. P. Serviços de pertinência para clusters de alta disponibilidade. **Universidade de São Paulo**, 2004.

FIRMIN, F. **The Evolved Packet Core**. 2017. Disponível em: <http://www.3gpp.org/technologies/keywords-acronyms/100-the-evolved-packet-core>. Acesso em: 12.8.2017.

GARTNER, F. C. **Fundamentals of fault tolerant distruted computing in asynchronous environments**. Departament of Computer Science. Darmstadt University of Technology: [s.n.], 1967.

GRUB, P.; TAKANG, A. A. **Software Maintenance - Concepts and Practice**. [S.l.]: World Scientific Publishing, 2003.

GURUGé, A. **Web services - Theory and Practice**. [S.l.]: Elsevier, 2004.

HOCHSTETLER, S.; BERINGER, B. **Linux Clustering with CSM and GPFS**. [S.l.]: IBM Redbooks, 2004.

JAYASWAL, K. **Administering Data Centers: Servers, Storage, and Voice over IP**. Indianapolis, IN 46256: Wiley, 2005.

MAGUELA, L. M.; AQUINO, G. P. **Segurança na rede LTE**. 2015. Disponível em: <http://www.inatel.br/biblioteca/pos-seminarios/seminario-de-redes-e-sistemas-de-telecomunicacoes/iii-srst/9477-seguranca-na-rede-lte>. Acesso em: 01.10.2017.

MARSHALL, G.; CHAPMAN, D. **Resilience, Reliability and Redundancy**. 2002. Copper Development Association. Disponível em: <http://admin.copperalliance.eu/docs/librariesprovider5/power-quality-and-utilisation-guide/41-resilience-reliability-and-redundancy.pdf?sfvrsn=4&sfvrsn=4>. Acesso em: 12.10.2017.

MAURO, D. R.; SCHMIDT, K. J. **SNMP Essencial**. [S.l.]: O'REILLY, 2001.

MOSER, S. L. **Alta disponibilidade: Um estudo de caso em um ambiente de imagem única de produção**. Florianópolis: [s.n.], 2004.

OLSSON, M. et al. **EPC and 4G Packets Networks - Driving the Mobile Broadband Revolution**. [S.l.]: Elsevier, 2013.

SCHMIDT, L. B. **Estudo da confiabilidade em sistemas série-paralelo com dois modos de falha**. 2017. 15 p. Dissertação (Pós-Graduação em Engenharia de Produção) — Universidade Federal do Rio Grande do Sul, Viçosa, 2017.

SKINNER, T. **Global LTE subscribers double to 1.29 billion – GSA**. 2016. Disponível em: <http://telecoms.com/473136/global-lte-subscribers-double-to-1-29-billion-gsa/>. Acesso em: 01.10.2017.

SOUZA, G. C.; PIRES, G. P.; LIMA, L. G. de. O protocolo diameter aplicado em redes lte. **Instituto Nacional de Telecomunicações**, 2015.

TEODORO, G. et al. Load balancing on stateful clustered web servers. **Universidade Federal de Minas Gerais**, 2004.

W3C. **Web Services Glossary**. 2004. Disponível em: <http://www.w3.org/TR/ws-gloss>. Acesso em: 28.8.2017.

WEBER, T. S. Um roteiro para exploração dos conceitos básicos de tolerância a falhas. **Apostila do Programa de Pós-Graduação–Instituto de Informática-UFRGS. Porto Alegre**, 2003.