UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS QUIXADÁ
BACHARELADO EM SISTEMAS DE INFORMAÇÃO

ROBERTO CABRAL RABÊLO FILHO

MODELAGEM E IMPLEMENTAÇÃO DO SOFTWARE DE DNS DINÂMICO

QUIXADÁ
2012

ROBERTO CABRAL RABÊLO FILHO

MODELAGEM E IMPLEMENTAÇÃO DO SOFTWARE DE DNS DINÂMICO

Trabalho de Conclusão de Curso submetido à Coordenação do Curso de Graduação em Sistemas de Informação da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel.

Área de concentração: computação

Orientadora Prof. Arthur de Castro Callado

QUIXADÁ
2012

Dados Internacionais de Catalogação na
Publicação
Universidade Federal do
Ceará
Biblioteca do Campus de
Quixadá

R144m Rabêlo Filho, Roberto Cabral.
Modelagem e implementação do Software de DNS dinâmico / Roberto Cabral Rabêlo Filho. – 2012.
36 f. : il. color., enc. ; 30 cm.

Monografia (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Sistemas de Informação, Quixadá, 2012.
Orientação: Prof. Dr. Arthur de Castro Callado.

1. Nomes de domínio na internet. 2. Internet 3. Redes de Computadores. I. Título.

CDD 005.432

ROBERTO CABRAL RABÊLO FILHO

MODELAGEM E IMPLEMENTAÇÃO DO SOFTWARE DE DNS DINÂMICO

Trabalho de Conclusão de Curso submetido à Coordenação do Curso de Graduação em Sistemas de Informação da Universidade Federal do Ceará como requisito parcial para obtenção do grau de Bacharel.

Área de concentração: computação

Aprovado em: _____ / junho / 2012.

BANCA EXAMINADORA

_____
Prof. Dr. Arthur de Castro Callado
Universidade Federal do Ceará-UFC

_____
Prof. Msc. Marcos Dantas Ortiz
Universidade Federal do Ceará-UFC

_____
Prof. Msc. Marcio Espíndola Freire Maia
Universidade Federal do Ceará-UFC

Aos meus pais...

AGRADECIMENTOS

Primeiramente gostaria de agradecer a Deus por me dar força diariamente para continuar.

A toda minha família que sempre esteve ao meu lado dando o apoio necessário para eu seguir em frente, em especial minha mãe Rosângela a qual nunca vou conseguir achar palavras para defini-la, meu pai Roberto que contribuiu muito com meu aprendizado, não conheço outro pai que passasse a tarde ensinando uma criança que não conseguia nem distinguir um “v” de um “f”, e aos meus irmãos Ana e Robert, que sempre estão do meu lado.

A minha namorada Leonara, que sempre esteve ao meu lado me apoiando e conseguindo me aturar.

Ao professor Arthur, que contribuiu muito para minha formação acadêmica nesses dois anos de orientação.

A todos os professores que contribuíram de forma direta e indireta na minha formação acadêmica e pessoal.

E a todas as outras pessoas que não foram aqui citadas, mas contribuíram de alguma forma para o desenvolvimento desse trabalho.

"O único lugar onde o sucesso vem
antes do trabalho é no dicionário."
(Albert Einstein)

RESUMO

Esse trabalho pretende disponibilizar para a sociedade um serviço dinâmico de nomes de domínio (*Dynamic Domain Name System - DDNS*) de propósito geral. Esse software é livre, de código aberto e viabilizará o acompanhamento da macromobilidade dos usuários, isto é, saber em quais redes ele entrou e quando. O nosso sistema ficou dividido em três partes principais que são o servidor intermediário, que é uma Interface de Gateway Comum (*Comon Gateway Interface - CGI*) implementado em C e é quem faz a comunicação entre o cliente e o servidor DNS; o programa cliente, que é um programa desenvolvido em Java e é o responsável pela atualização dos dados no servidor intermediário; e o servidor Apache, que intermedeia a comunicação entre o programa cliente e o servidor intermediário.

Palavras chave: DNS. DDNS. Macromobilidade.

ABSTRACT

This paper aims to provide a general-purpose Dynamic Domain Name System (DDNS). This software is free, open source and enables the monitoring the user macro mobility, i.e., to know which networks he has entered and when. Our system was divided into three main parts that are the intermediate server, that is a Common Gateway interface (CGI), implemented in C and makes communication between the client and a DNS server; the client program, that is a program developed in Java and the responsible for updating the data in the intermediate server; and an Apache server, that makes the communication between client program and the intermediary server.

Keywords: DNS. DDNS. Macro Mobility.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

Esse trabalho faz parte de um projeto maior intitulado Interligação e Mobilidade de Usuários VoIP no Ceará que vem sendo desenvolvido pela Universidade Federal do Ceará - Campus de Quixadá, financiado pela FUNCAP (Fundação Cearense de Apoio ao Desenvolvimento Científico e Tecnológico) (CALLADO, 2010). O referido projeto tem o intuito de lidar com três frentes: prover a infraestrutura telefônica de VoIP necessária ao Campus da UFC em Quixadá; realizar a interligação dessa infraestrutura com a infraestrutura do campus de Sobral através dos projetos fone@RNP (fone@RNP,2011) e VoIP4all (VoIP4all,2011); criar um arcabouço tecnológico de acompanhamento de usuário móvel, desenvolvendo um serviço dinâmico de nomes de domínio (*Dynamic Domain Name System* – DDNS) livre cujo código será aberto.

Nesse trabalho pretende-se detalhar os passos para o desenvolvimento de um servidor dinâmico de nomes de domínio (DDNS) livre de código aberto a ser desenvolvido com o intuito de agregar a capacidade de acompanhamento de macromobilidade dos usuários ao software de ramal virtual usado no serviço fone@RNP. E também permitir que empresas responsáveis pela sua faixa IP, com esse *software*, possam implementar o serviço de DNS reverso.

Há vários serviços com versões gratuitas que podem ser usados para fazer o controle de DNS dinâmico, mas não conseguimos encontrar algum que faça o acompanhamento da mobilidade dos usuários. Todos os que encontramos, como o *no-ip* (NO-IP, 2012) e o superDNS (SUPERDNS, 2012), guardam apenas o endereço de IP atual do *host* (computador) inviabilizando assim que seja feito o acompanhamento da macromobilidade dos usuários (porque não guardam o histórico) e não permitindo que os respectivos provedores responsáveis pela sua faixa de IP usem do serviço de DNS reverso. Já o sistema que desenvolvemos mantém um *log* de todas as atualizações de endereço IP permitindo assim que seja possível acompanhar a macromobilidade dos usuários.

O sistema que foi desenvolvido é livre e de código aberto e está disponibilizado para a sociedade na forma de um software que pode ser usado tanto por usuários comuns (residenciais) como por empresas. O mesmo está disponível no *Google Code* e pode ser encontrado no *link* a seguir: http://dns-dinamico-ufc-quixada.googlecode.com/svn/trunk/.

As empresas, ao implantá-lo, terão acesso aos dados internos do sistema. Assim vão poder fazer um acompanhamento dos usuários na rede, além de poder ter o serviço de DNS reverso (caso seja responsável administrativa pela faixa de IPs que utiliza).

Segundo o Ibope (2011), em uma pesquisa feita no segundo trimestre de 2011, o Brasil atingiu 77,8 milhões de pessoas que acessam internet, 5,5% a mais que o segundo trimestre de 2010 e 20% que o segundo trimestre de 2009. Se levarmos em conta apenas os usuários ativos em residências, o Brasil chegou a 37 milhões em agosto de 2011, o que significa um crescimento de 14,4% em um ano. A necessidade de registro de domínios dinâmicos aumenta com o crescimento do número de usuários, já que mais pessoas precisarão oferecer serviços em suas máquinas para outras pessoas na rede. O endereço IP que é disponibilizado para a maioria das residências pelos provedores de internet não é um IP fixo e sim um IP dinâmico. Isso limita algumas ações de usuário, como o acesso remoto ao próprio *host*, ou a utilização do *host* como um servidor de arquivos ou servidor *Web*. Para conseguir junto ao provedor de internet um IP fixo o custo é mais elevado que o custo de um IP dinâmico e muitos provedores só o disponibilizam para empresas. O nosso sistema pretende viabilizar que um usuário comum possa prover esses serviços possuindo apenas um IP dinâmico.

2 REFERENCIAL TEÓRICO

Nesta sessão iremos abordar três tópicos que estão diretamente relacionados a esse trabalho, começaremos com DNS que é a espinha dorsal desse trabalho já que nos propomos a desenvolver um serviço de DDNS. Em seguida falaremos um pouco sobre mobilidade já que nosso sistema irá gerar dados que viabilizarão o acompanhamento da macromobilidade dos usuários e fecharemos a sessão com servidores *web* que irão intermediar a comunicação do nosso *software* cliente com o *software* servidor.

2.1 DNS

Os *hosts* conectados à internet são identificados por um endereço único chamado por endereço IP. Para entendermos melhor como se dá essa identificação podemos fazer uma analogia com a nossa sociedade da seguinte forma: quando fazemos uma compra pela internet de um livro, por exemplo, esperamos receber esse livro em nossa casa. Para isso, disponibilizamos para a empresa responsável pelo envio o nosso endereço que é único e

composto pelo nome da rua, número, bairro, CEP, cidade. Com esses dados, a empresa pode colocar nossa encomenda nos correios para que seja entregue exatamente no endereço especificado.

Na internet, essa comunicação acontece de maneira semelhante. Para que um *host* seja identificado ele necessita de um endereço único, o seu endereço IP com 32 bits (equivalente a 4 bytes) em sua versão 4 (RFC 791), ou 128 bits (16 bytes) em sua versão 6 (RFC 2460; RFC 4291). O endereço IP tem uma estrutura hierárquica rígida, e "ao examiná-lo da esquerda para a direita, obtemos gradativamente mais informações específicas sobre onde o hospedeiro está localizado na internet" (KUROSE; ROSS, 2010, p.96).

No entanto, é difícil imaginar como seria a internet hoje em dia se para comunicarmo-nos com um site tivéssemos que digitar seu endereço IP. Seria quase impossível lembrarmos os endereços IPs de todos os sites que acessamos. Por exemplo, se fôssemos acessar o site do Google iríamos identificá-lo como http://74.125.234.80/ seu endereço IP, e não como http://www.google.com.br, seu nome de hospedeiro. Este último é bem mais fácil de ser lembrado, mas, em contrapartida, é bem mais difícil de ser processado pelos roteadores que utilizam o endereço IP para realizar esse endereçamento.

Para que seja possível conciliarmos essas duas formas de endereçamento, precisamos dispor de um serviço de diretório que traduza o nome de hospedeiro para o seu endereço IP. Essa é a principal tarefa do *domain name system* (DNS) (RFC 1034; RFC 1035) que segundo (KUROSE; ROSS, 2010) é um banco de dados distribuído implementado em uma hierarquia de servidores de nome (servidores DNS), e um protocolo da camada de aplicação que permite que hospedeiros consultem o banco de dados distribuído. Vejamos agora o passo-a-passo que acontece quando colocamos, em um navegador de internet, a URL http://www.google.com.br. Primeiro, o navegador extrai o nome de hospedeiro www.google.com.br da URL e passa esse nome para o lado cliente da aplicação DNS em nossa máquina. Em seguida, este cliente DNS envia uma consulta para um servidor DNS contendo o nome do hospedeiro e aguarda informação do endereço IP correspondente. Quando o cliente DNS recebe uma resposta, já pode abrir uma conexão TCP com o endereço IP 74.125.234.80 recebido. A Figura 1 ilustra esse passo-a-passo.

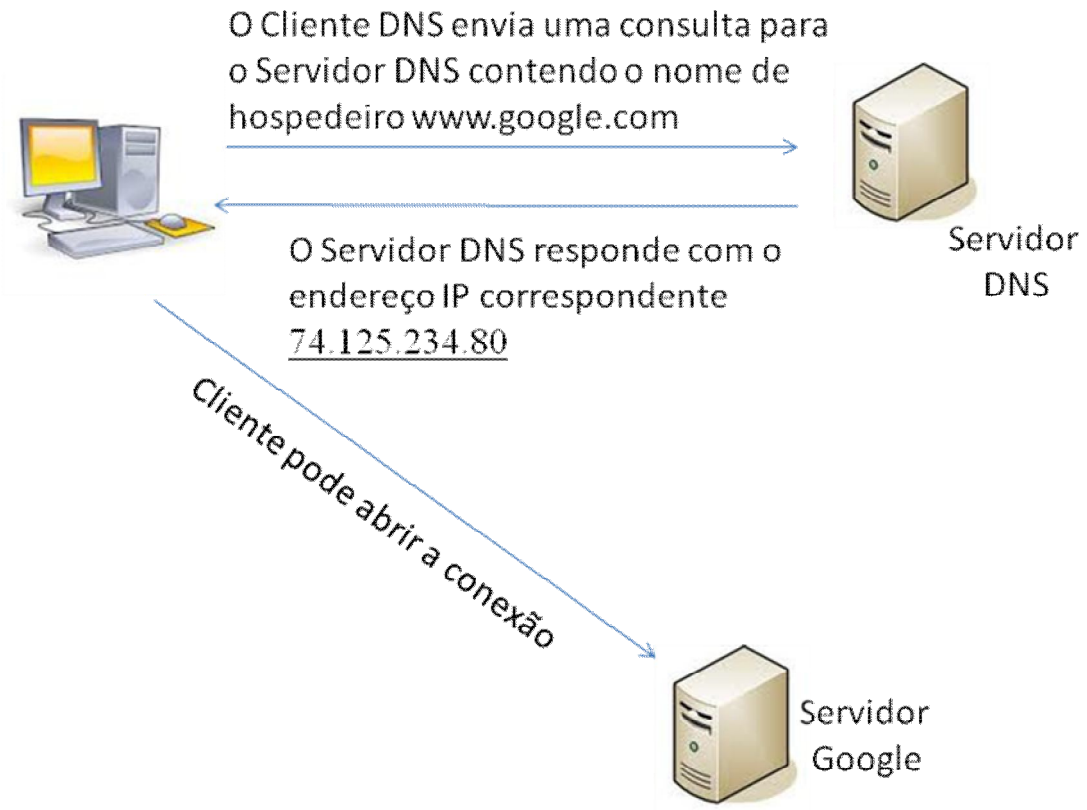

Figura 1- Funcionamento do DNS (produzida pelo autor)

Fazendo o papel inverso do DNS tradicional temos o DNS reverso que a partir de um endereço IP busca o nome do endereço de domínio dessa máquina. Segundo (MORIMOTO, 2009) ele é o recurso que permite que outros servidores verifiquem a autenticidade do seu servidor, checando se o endereço IP atual bate com o endereço IP informado pelo servidor DNS. O DNS reverso é usado principalmente por servidores de email para verificar se o IP da máquina que enviou o email corresponde ao IP do remetente, pois qualquer pessoa pode enviar emails alterando o campo remetente. Um problema que ocorre se a empresa não configurar o DNS para responder as checagens de DNS reverso é que os servidores de email não vão conseguir verificar a autenticidade dos emails dessa empresa, logo eles vão ser classificados como possível *spam*[1].

Vimos que o IP em sua versão 4 possui um comprimento de 32 bits. Logo, existem 2^{32} endereços IP possíveis, número esse que chega a pouco mais de 4 bilhões de endereços. A internet, rede mundial de computadores é composta pele junção de todos esses

[1] Spam é o termo usado para referir-se aos *e-mails* não solicitados, que geralmente são enviados para um grande número de pessoas.

endereços. Assim ficaria quase impossível gerenciar uma rede desse porte caso a mesma não fosse subdividida em sub-redes, imagine quão custoso seria o envio de dados em "*broadcasts*".

Já sabemos que existem várias sub-redes derivadas de uma rede maior, internet, e que para usar a internet temos que ter um endereço IP, mas como esse endereço IP chega até as pessoas? Para que uma pessoa possa receber um endereço IP ela tem que solicitá-lo a um provedor de internet. Os provedores de internet por sua vez solicitam blocos de endereços IP à autoridade responsável pela administração dos mesmos. "Endereços IP são administrados sob a autoridade da *Internet Corporation for Assigned Names and Numbers* (ICANN) (ICANN, 2011), com base em diretrizes estabelecidas na RFC 2050" (KUROSE; ROSS, 2010, p. 256). O provedor de internet com posse de um bloco de endereços IP já é capaz de distribuí-los para seus clientes. Segundo (KUROSE; ROSS, 2010) geralmente é usado o Protocolo de Configuração Dinâmica de Hospedeiros (DHCP) (RFC 2131) para realizar essa distribuição de endereços IP. O DHCP Permite que o endereço IP seja distribuído de duas maneiras distintas: endereços fixos e dinâmicos. No primeiro, ele aloca um endereço IP para o host e sempre que ele for se conectar à internet ele vai receber o mesmo endereço já no segundo, o endereço IP alocado é temporário e muda sempre que o host conectar-se à internet. Existem vantagens e desvantagens nessas duas maneiras de distribuição de IP citadas.

Com o IP fixo, o usuário vai sempre saber qual o endereço que pertence ao seu host, podendo assim ter acesso remoto a ele e também usá-lo como servidor de arquivos ou servidor web. No entanto, um IP fixo é bem mais custoso para os provedores de internet que o IP dinâmico e consequentemente mais caro para os clientes. Isso ocorre porque quando um provedor aloca um endereço IP fixo para uma pessoa esse endereço não pode ser alocado para mais ninguém e se ele alocasse um IP fixo para todos seus clientes seria um desperdício, porque vimos que a quantidade de endereços IPv4 é bastante limitada e o provedor de internet possui muitos clientes que não vão estar conectados ao mesmo tempo. Por esse motivo, os provedores de internet optam em disponibilizar para a maioria de seus clientes um endereço IP dinâmico, pois ele elimina esse problema de ociosidade dos endereços, diminuindo assim os custos para os provedores e para os usuários.

O IP fixo, frequentemente, é disponibilizado pelos provedores de internet apenas para empresas ou ainda para alguns usuários domésticos mediante pagamento de uma taxa (normalmente cara, frente ao valor da conexão). Mas os usuários comuns podem acessar seus hosts remotamente mesmo possuindo apenas IP dinâmico? A resposta para essa pergunta é

“sim”, eles podem ter acesso remoto aos seus *hosts* usando o serviço de DDNS, serviço esse que segundo (ZHAOHUI, L; JIANING, L, 2009) se baseia no DNS tradicional e fornece um mecanismo de atualização que possibilita a utilização de nomes de domínio na internet mesmo usando endereços de IP dinâmicos. Ele funciona da seguinte forma:

1. O usuário cria um nome de domínio e associa esse nome de domínio ao seu endereço IP;
2. O usuário instala um pequeno software em sua máquina que ficará responsável em sempre que o endereço IP sofrer alguma alteração enviar uma mensagem para o servidor informando que o endereço IP desse nome de domínio foi alterado;
3. Agora esse nome de domínio fica associado ao endereço IP correspondente e quando for necessário abrir uma conexão com essa máquina, basta usar o nome de domínio, pois o mesmo sempre vai estar atualizado com o endereço IP da máquina no momento atual.

A Figura 2 ilustra o funcionamento do DNS dinâmico onde o *host* “host-teste.com” informa seu endereço IP ao servidor de DDNS sempre que seu endereço IP é alterado. Assim quando o usuário quiser se comunicar com esse *host* ele irá pegar o endereço IP com o servidor DDNS e depois já conseguirá abrir uma conexão direta com o *host* solicitado.

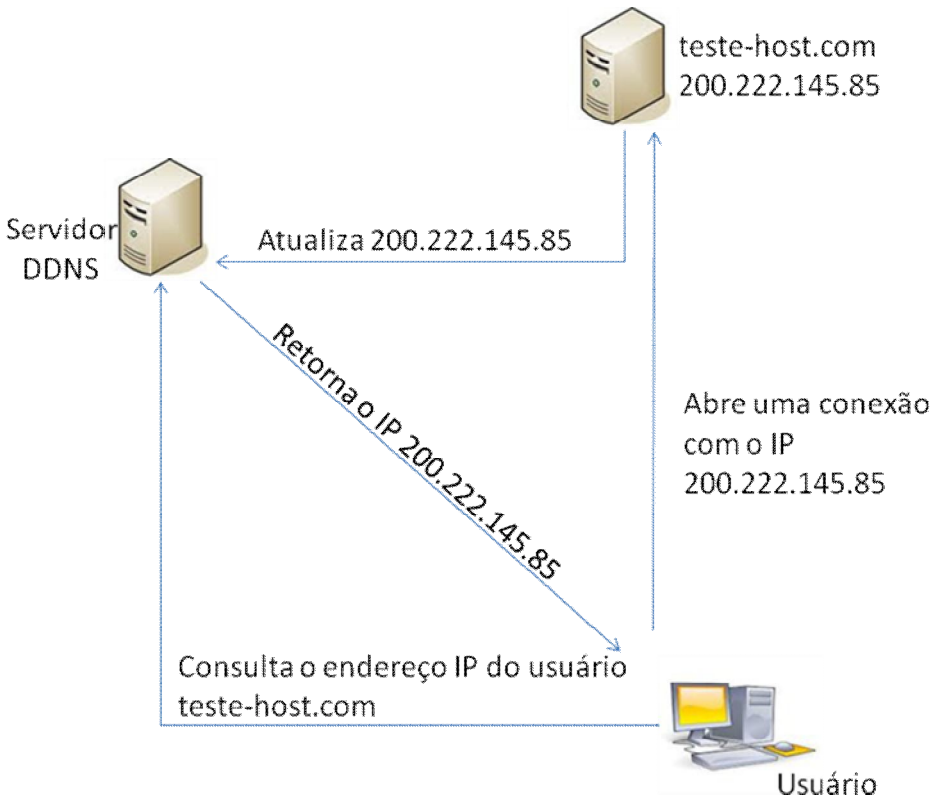

Figura 2- Funcionamento do DNS dinâmico (produzida pelo autor)

No desenvolvimento deste trabalho analisamos dois cenários distintos, no primeiro o cliente fazia suas requisições diretamente ao servidor DNS, sem a necessidade de um intermediário, e no segundo um programa que trata essas requisições antes de elas serem enviadas ao servidor. Depois de analisarmos esses dois cenários e vimos que no primeiro todas as requisições de atualização iriam diretamente ao servidor DNS, assim sempre teríamos que reescrever o arquivo de configuração DNS. Esse arquivo é onde estão contidos os nomes de domínio dos *hosts* com seus respectivos endereços IP. Depois disso bastaria reinicializar o servidor. Porém nem sempre o endereço IP terá mudado. Assim este processamento estaria sendo desperdiçado. Por isso decidimos usar o segundo cenário, onde primeiro essas requisições passam por um programa, que no caso é nosso servidor intermediário, e só passarão para servidor DNS as requisições necessárias.

Uma característica interessante de se usar o serviço DDNS é que o usuário pode não somente mudar de endereço como também mudar de rede e continuar atendendo ao serviço com o mesmo nome, isto é, dá mobilidade ao serviço.

2.2 MOBILIDADE

Segundo (KUROSE; ROSS, 2010) um usuário fisicamente móvel pode possuir ou não mobilidade de rede, em três possíveis cenários distintos que podem ser vistos na Figura 3:

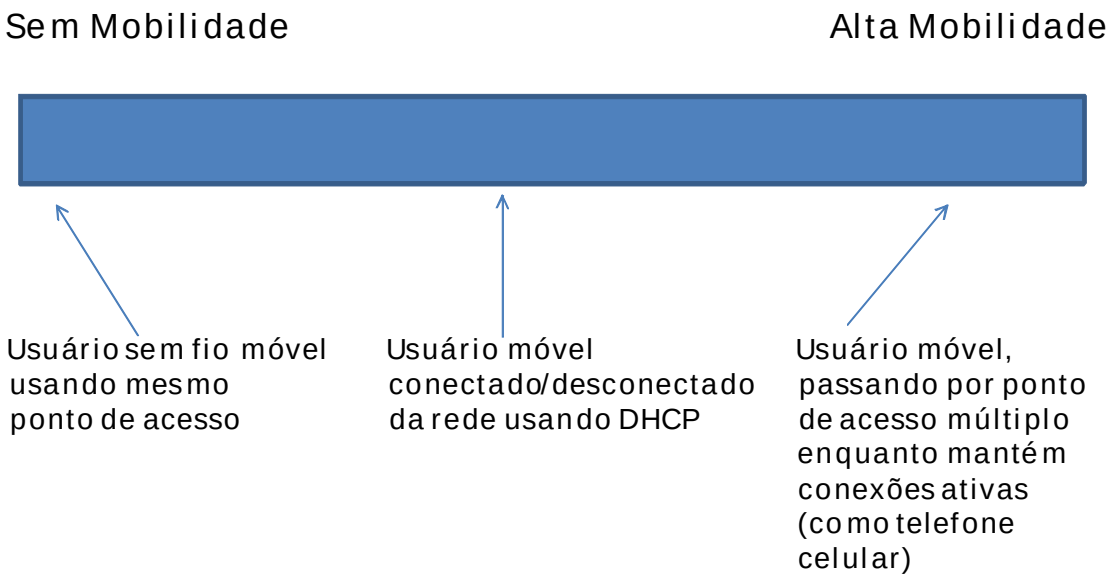

Figura 3 Cenário de Mobilidade. Fonte: adaptado de (KUROSE; ROSS, 2010, p.406).

O usuário que se encontra mais à esquerda na figura, que pode ser uma pessoa com um *notebook* se locomovendo em sua casa conectada a uma mesma rede de acesso sem fio, por exemplo, não é um usuário móvel da perspectiva da camada de rede. Já o usuário que se encontra no meio, que pode ser um usuário que usa seu *notebook* para se conectar a uma rede em seu trabalho e usa o mesmo *notebook* para conectar-se a uma rede doméstica em sua casa, por exemplo, pode ser considerado um usuário móvel mesmo não possuindo o mesmo grau de mobilidade do usuário que está mais a direita da figura, usuário esse que consegue se movimentar entre várias redes mantendo suas conexões ativas, por exemplo, um usuário falando ao telefone dentro de taxi via VoIP, ele transita em diferentes redes, mas sem descontinuidade de seu sinal.

Esse trabalho não suportará o acompanhamento de usuários com alta mobilidade, pois nossa atualização não é instantânea gerando assim uma descontinuidade do serviço o que pode acontecer com esse tipo de usuário. Por sua vez, o usuário mais a esquerda, que é um usuário sem nenhuma mobilidade, poderá usar o sistema mas não poderemos acompanhar a

mobilidade dele. Isso ocorre porque ele não possui mobilidade já que sempre usa o mesmo ponto de acesso. Neste trabalho forneceremos uma ferramenta que viabiliza o acompanhamento da mobilidade apenas desse usuário intermediário que é um usuário móvel transitando entre redes, mas com a descontinuidade do serviço durante essa transição. Um exemplo desse tipo de usuário é um usuário que usa o serviço fone@RNP em seu *notebook*.

Segundo (fone@RNP, 2011) o serviço fone@RNP é um serviço que permite o encaminhamento de chamadas, ou seja, a interconexão das instituição clientes da rede VoIP da RNP. Segundo (CALLADO, 2010) os usuários acessam esse serviço não apenas por ramais fixos, mas também por ramais virtuais que podem ser usados em qualquer dispositivo com capacidade sonora conectado à internet através da adição de um simples software. Os usuários de ramais virtuais em sua grande maioria se enquadram nesse segundo cenário, porque eles podem possuir mobilidade, por poderem ser instalados em dispositivos móveis, e seu endereço IP é alterado quando mudam de rede.

2.3 SERVIDOR WEB

Segundo (KUROSE; ROSS, 2010) uma aplicação de rede consiste em pares de processos que enviam mensagens uns para os outros por meio de uma rede. Uma das arquiteturas mais usadas na comunicação em rede é a arquitetura cliente servidor, onde “os processos clientes interagem com processos servidores, localizados em distintos computadores hospedeiros, para acessar os recursos compartilhados que estes gerenciam” (COULOURIS; DOLLIMORE; KINDBERG, 2007, p.43).

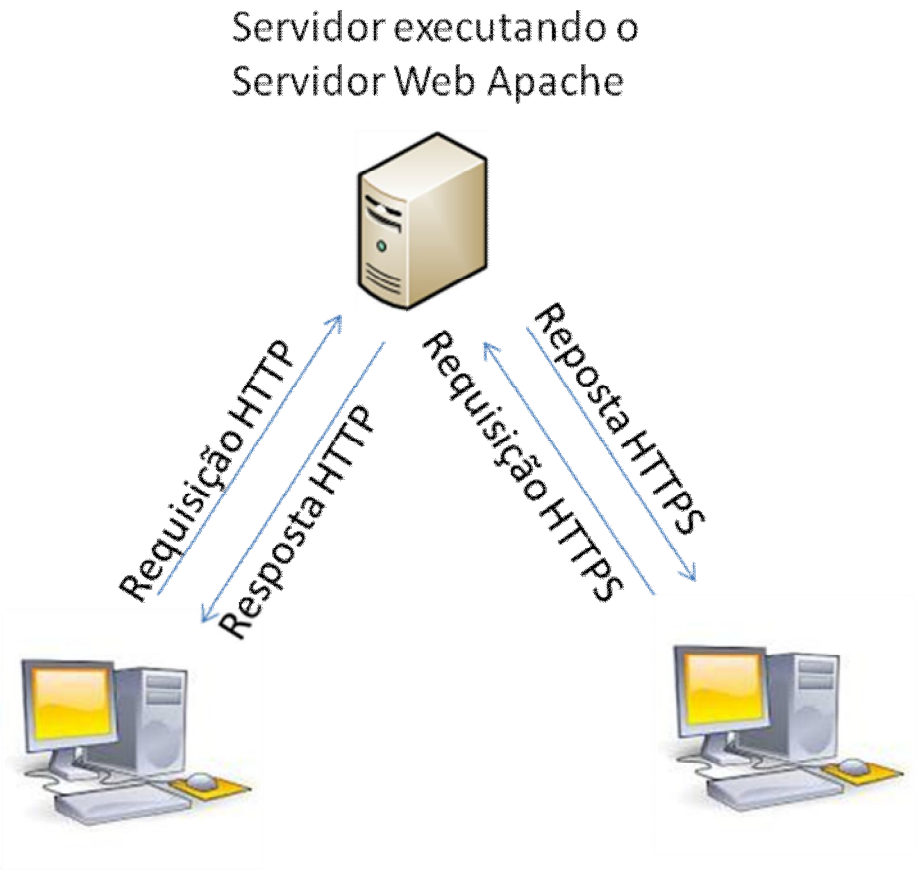

Figura 4 - Comportamento de requisição-resposta do HTTP. Fonte: adaptado de (KUROSE; ROSS, 2010, p.73).

Na Figura 4 podemos perceber melhor como funciona a arquitetura cliente servidor. Temos o cliente que faz uma requisição ao servidor e esse servidor por sua vez irá tratar a requisição do cliente podendo enviar uma resposta.

Ao acessar uma página na Internet, o pedido de acesso é uma mensagem enviada a um servidor web. Para intermediar essa comunicação temos o HTTP – Protocolo de Transferência de Hipertexto (*Hypertext Transfer Protocol*) (RFC 1945; RFC 2616) que segundo (KUROSE; ROSS, 2010) define a estrutura e a forma como essas mensagens são trocadas entre o servidor e o cliente.

O HTTP é implementado por dois programas diferentes, o programa cliente que é implementado na máquina cliente e o programa servidor que é implantado no servidor. Ambos comunicam-se por meio de troca de mensagens. A Figura 4 ilustra o comportamento de uma requisição-resposta HTTP, onde temos um servidor executando o servidor *web apache* que recebe requisições HTTP do cliente, trata essas requisições e envia a resposta

também por meio do HTTP. É importante lembrar que o HTTP usa o TCP como seu protocolo de transporte, não precisando assim se preocupar como dados perdidos ou controle de banda que são de responsabilidade do TCP.

Segundo (COULOURIS; DOLLIMORE; KINDBERG, 2007) os programas executados pelos servidores web para gerar conteúdo para seus clientes são frequentemente denominados como programas CGI – Interface de *Gateway* Comum (*Comon Gateway Interface*). Um programa CGI pode ter qualquer funcionalidade específica do aplicativo, ele analisa argumentos fornecidos pelo cliente (podem vir na URL pelo método GET ou no corpo da mensagem pelo método POST do protocolo HTTP) e produz o conteúdo do tipo solicitado. A CGI não é uma linguagem de programação, assim pode ser usada qualquer linguagem para escrever um programa CGI, desde que a mesma tenha suporte para leitura e escrita de dados.

O protocolo HTTP não é aconselhado para ser usado quando estiver sendo transmitindo dados sensíveis pela rede, pois suas mensagens são trocadas abertamente pelo meio de comunicação, assim facilmente uma pessoa mal intencionada pode capturar as mensagens usando um programa *sniffer* e ter acesso às informações lá contidas. Para resolver esse problema de segurança usamos o HTTPS que é a combinação do protocolo HTTP com a camada de segurança de sockets (SSL – *Secure Sockets Layer*). O SSL atua entre a camada de aplicação e a camada de transporte, segundo (KUROSE; ROSS, 2010) ele aprimora o TCP com sigilo, integridade dos dados, autenticação do servidor e autenticação do cliente.

O SSL é dividido em três fases, que são: apresentação (*handshake*), derivação de chave e transferência de dados. Na fase de apresentação é necessário estabelecer uma conexão TCP entre os dois comunicantes, ter certeza que a pessoa com a qual você está se comunicando é a pessoa com quem você queria se comunicar. Essa identificação é feita a partir de um certificado digital que é assinado por uma autoridade certificadora. Usando esse certificado, cada um dos comunicantes envia ao outro uma chave mestra que será usada para a criação das outras chaves necessárias à comunicação e servirá apenas para essa comunicação. Essa chave será enviada pela rede criptografada com a chave publica da outra pessoa garantindo assim que apenas os dois comunicantes saberão qual é a chave mestra. Segundo (KUROSE; ROSS, 2010) a chave mestra que está compartilhada entre os dois comunicantes poderia ser usada como chave de sessão simétrica, mas por questão de segurança é mais interessante cada um usar individualmente chaves criptográficas diferentes, bem como chaves diferentes para criptografia e verificação da integridade. Desse modo, eles irão usar a chave

mestra para criar quatro chaves que serão compartilhadas entre os dois comunicantes. Feito isso já é possível haver uma transferência de dados de forma segura.

3 PROCEDIMENTOS METODOLÓGICOS

O sistema foi desenvolvido de forma modular, visando dar uma maior legibilidade e facilidade na manutenção. Em seu desenvolvimento usamos o modelo de desenvolvimento cascata, no qual as fases de desenvolvimento são realizadas sequencialmente.

Começamos nosso desenvolvimento com a fase de levantamento de requisitos na qual coletamos e analisamos os dados. A partir desse levantamento identificamos os seguintes requisitos funcionais, não funcionais e de regras de negócio, que foram definidos da seguinte forma.

Requisitos Funcionais:

- O sistema deve fornecer a autenticação de usuários;
- O sistema deve fornecer o cadastro e remoção de hosts;
- O sistema deve monitorar o IP da máquina e atualizar esse IP sempre que houver alguma mudança.

Requisitos de Regra de Negócio:

- Sempre que o usuário se autenticar no sistema deveremos verificar o seu IP; se tiver mudado, será atualizado; caso contrário, atualizamos apenas o horário;
- O nome dos hosts deverá ser único;
- A autenticação será feita por email e senha.

Requisitos Não Funcionais:

- O sistema deve se mostrar eficiente;
- O sistema deve ter uma boa disponibilidade.

Depois do levantamento de requisitos começamos a fase de projeto do software. Esta foi a fase onde pensamos como o nosso sistema estaria organizado e nessa fase foi produzido o fluxograma que pode ser visto na Figura 5.

A fase seguinte foi a fase de implementação. Nessa fase foram implementados tanto o servidor intermediário como o programa cliente. Mais detalhes sobre essa fase podem ser vistos na sessão a seguir.

Para testar o sistema fizemos testes unitários tanto no servidor intermediário quanto no programa cliente e depois desses testes foi feito o teste de integração onde os dois programas se comunicavam entre si por intermédio do servidor Apache.

4 DESENVOLVIMENTO

O sistema está dividido em três partes, o servidor intermediário, o servidor *Web* e programa cliente. O servidor intermediário é o cérebro do nosso sistema. É dele a responsabilidade por intermediar a comunicação entre o cliente e o servidor DNS além de fazer toda a comunicação com o banco de dados e possuir todas as regras de negócio do sistema. O programa cliente, por sua vez, é a interface visível ao usuário e por onde ele realiza todas suas ações com o sistema. Para intermediar a comunicação entre o programa cliente e o nosso servidor intermediário e ainda facilitar o desenvolvimento do sistema temos um servidor *Web*.

Nosso banco de dados é acessado diretamente pelo nosso servidor intermediário e possui quatro tabelas que são: tabela tipo usuário, tabela usuário, tabela host e tabela atualização. Na tabela tipo usuário temos os tipos de usuário suportados pelo sistema, nesta versão temos apenas três tipos usuários que são alunos, professores e administrador. Na tabela usuário teremos os dados dos usuários do sistema, o arquivo de configuração do banco de dados pode ser encontrado no APÊNDICE A – arquivo de configuração do banco de dados. O servidor intermediário também tem acesso ao arquivo de configuração do DNS que é onde estão os nomes de domínio dos *hosts* e seus respectivos endereços IP.

O fluxograma a seguir ilustra o funcionamento do sistema.

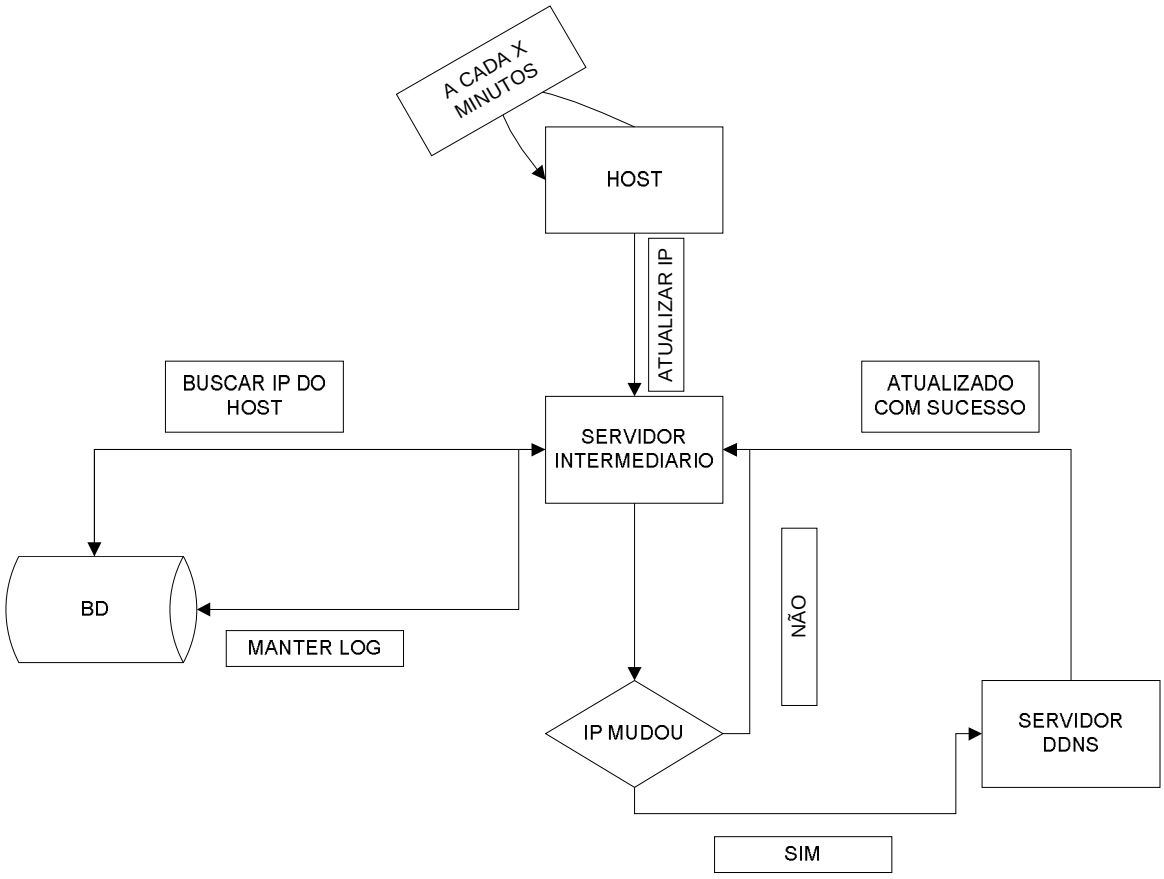

Figura 5 – Fluxograma de Atualização de Hosts

Nesse fluxograma temos o *host*, que é a máquina onde está instalado o programa cliente, esse *host* fica enviando de tempos em tempos uma requisição de atualização para o servidor intermediário, optamos por fazer essa atualização de tempos em tempos e não apenas quando o endereço IP sofrer alterações para termos informações mais reais sobre a mobilidade do usuário, pois um usuário que passou quinze dias sem entrar na rede com esse host é diferente de um usuário que passou quinze dias conectado com o mesmo IP.

Recebendo essa requisição, o servidor intermediário buscará no banco de dados o endereço IP deste *host* e verificará se o mesmo foi alterado. Caso ele tenha sido alterado o servidor intermediário atualizará os dados no banco, depois ele reescreverá o arquivo de configuração do DNS, reiniciará o serviço BIND e manterá o *log* de atualização no banco. Mas se o endereço IP não tiver sofrido alteração será apenas mantido o *log* de atualização no banco de dados.

No desenvolvimento do sistema começamos usando o TDD, mas essa técnica estava deixando o desenvolvimento muito lento e se continuássemos a utilizá-la não teríamos

conseguido desenvolver todo o projeto a tempo hábil, assim continuamos o desenvolvimento sem TDD.

O servidor intermediário é um programa CGI que foi desenvolvido na linguagem C e é o responsável por tratar as requisições do usuário que são feitas pelo programa cliente que foi desenvolvido em Java. A comunicação entre ele o programa cliente se dá por meio de requisições HTTP ou HTTPS. Também são de responsabilidade do servidor intermediário todas as comunicações com o banco de dados. Em nosso sistema optamos por usar o postgresql por ser um sistema gerenciador de banco de dados livre e código aberto.

Escolhemos fazer essa comunicação em rede por meio de um servidor web e escolhemos o Apache, porque assim não precisamos nos preocupar diretamente com transferência de dados sensíveis pela rede, pois o servidor Apache permite a utilização do HTTPS. Dessa forma, a mensagem transferida já vai criptografada evitando que uma pessoa mal intencionada tenha acesso a esses dados.

4.1 SERVIDOR INTERMEDIÁRIO

Neste tópico abordaremos a estrutura do servidor intermediário. Temos a seguir o detalhamento de todas as funções que estão presentes no servidor intermediário:

- char* getIp()

Essa função pega o IP do servidor que deve está no arquivo “conf” que é o arquivo de configuração do servidor intermediário.

- int enableRegister()

Essa função pega retorna “1” se o cadastro de novos usuários está ativo e “0” caso contrário. Esse deve está no arquivo “conf” que é o arquivo de configuração do servidor intermediário.

- char* searchIpHost(char * hostName)

Essa função recebe o nome do *host* e retorna o endereço IP do mesmo.

- int searchIdHost(char* hostName)

Essa função recebe o nome do *host* e retorna o ID do mesmo.

- int searchIdUser(char* email)

Essa função recebe o email do usuário e retorna o ID do mesmo.

- char* getValue(char *c,char *v)

Essa função recebe a string de consulta (*query string*) que é passada para pelo programa cliente para a CGI e o nome da variável que queremos buscar seu valor. Ela retorna NULL caso não exista essa variável na string de consulta ou o valor correspondente a essa variável.

- int validateIp(char *c)

Essa função recebe um endereço IP e verifica se ele é válido, retornando “1” caso seja válido e “0” caso contrário.

- char *getTime()

Essa função retorna a data e hora atual formatada no padrão que mantemos no banco.

- int getNat(char *hostName)

Essa função recebe o nome de um *host* e retorna “-1” se esse *host* não estiver cadastrado, “1” se quando esse *host* estiver sob NAT o sistema tiver que guardar se IP interno e “2” se for para guardar seu IP externo.

- int getTypeIp(char *hostName)

Essa função recebe o nome de um *host* e retorna “-1” se esse *host* não estiver cadastrado, “1” se esse *host* usa IPv4 e “2” se ele usa IPv6.

- int authenticateUser(char *email, char *password)

Essa função recebe o email e a senha de um usuário e faz a autenticação dele retornando “-2” se o usuário não estiver cadastrado, “1” se o usuário foi autenticado e “-1” caso contrário.

- int restartServer()

Essa função é chamada para reiniciar o servidor DNS, ela reescreve o arquivo de configuração do servidor DNS e chama um pequeno programa, também desenvolvido em C, que reiniciará o serviço de DNS, não reiniciamos o serviço dentro da própria função porque apenas o super usuário tem permissão para tal execução.

- int update(int idHost, int idUser, char *currentIp, char *lastIp, char *email, char *password)

Essa é uma das funções mais importantes do sistema, é de sua responsabilidade a atualização de um *host*. Ela recebe o ID do *host* e do usuário, o IP que o *host* tem no momento, o último endereço IP desse *host*, o email e a senha do usuário. Ela primeiro faz a validação desse usuário pelo seu email e sua senha, com o usuário validado ele verifica se seus IPs (antigo e atual) também são válidos. Com todos esses dados válidos ela verifica se o IP do usuário sofreu alterações, se não tiver sofrido alteração ela apenas guarda o log da atualização, mas se o IP tiver mudado ele será atualizado na tabela HOST chamará a função restartServer() e guardará o log da atualização.

- int addUser(char *email, char *password, char *userName, int userType)

Essa função recebe os dados de um novo usuário e se o sistema estiver ativo para novos cadastros de usuários ela faz o cadastro do mesmo retornando “-1” se esse email já foi cadastrado e “1” caso o cadastro tenha sido realizado com sucesso.

- int addHost(int userId, char *hostName, int nat, char *ip, int ipType, char *email, char *password)

Essa função recebe os dados de um novo *host*, o email e a senha do usuário que pretende cadastrar esse novo *host*. Primeiramente a função faz a autenticação do usuário e a validação do IP. Depois com o usuário e o IP validados ela insere o novo host retornando “-1” se esse *host* já foi cadastrado e “1” caso o cadastro tenha sido realizado com sucesso. É importante lembrar que o nome dos *hosts* é único, assim não podemos ter nomes repetidos mesmo sendo de usuários diferentes.

- void resultStatus(int result)

Sempre que é feito algum acesso ao banco de dados é chamada essa função passando o resultado desse acesso e ela mantém um log desse resultado no arquivo Servidor.log.

- int removeHost(int idHost, char *email, char *password)

Função responsável pela remoção de um *host.*

4.2 PROGRAMA CLIENTE

Neste tópico abordaremos a estrutura do programa que rodará na máquina do usuário. A seguir detalharemos as telas que serão visíveis para os usuários.

Figura 6 - Tela Escolher Servidor do Programa Cliente

Na Figura 6 temos a tela onde o usuário escolhe qual protocolo irá usar (HTTP ou HTTPS) e informa o endereço do servidor com o qual ele quer se comunicar, o programa tenta se comunicar com ele para saber se esse é um endereço válido se não conseguir se conectar é mostrada uma mensagem de erro para o usuário informando que o endereço é inválido. Conseguindo se conectar ele vai para a tela seguinte.

Figura 7 - Tela Escolher Serviço do Programa Cliente

Na tela da Figura 7 o usuário escolhe a opção que deseja executar no sistema, um usuário que ainda não é cadastrado tem a opção de se cadastrar em “Criar uma Conta”. Já usuários que são cadastrados poderão adicionar um novo host ou ativar atualização dinâmica.

Selecionando a opção criar conta será feita a verificação se o sistema está apto a aceitar novos cadastros. Caso ele não esteja aceitando novos cadastros essa informação é informada ao usuário a ele e continuará na tela da Figura 7, mas se o cadastro de novos usuários estiver ativo o usuário é redirecionado para a tela da Figura 8 onde poderá efetuar seu cadastro.

Figura 8 - Tela Cadastro Cliente do Programa Cliente

Caso o usuário selecione a opção cadastrar novo host ele irá ser redirecionado para a tela da Figura 9 onde ele irá informar os dados do host que pretende cadastrar e seus dados pessoais, depois de informar os dados ele clica em cadastrar e o sistema tentará realizar esse cadastro e lhe informará uma mensagem de sucesso e lhe redirecionará para a Figura 7 ou de falha lhe mantendo na mesma tela.

Figura 9 - Tela Cadastrar novo host do Programa Cliente

Na tela da Figura 12 o usuário ativa a atualização dinâmica de um *host*. Ele entra com os dados do *host* que vai ficar sendo atualizado dinamicamente e clica no botão atualizar, se os dados fornecidos para atualização estiverem corretos essa tela do sistema se fechará e aparecerá o ícone do sistema na barra de ferramentas de seu sistema operacional com pode ser visto na Figura 10. O sistema continuará executando e enquanto ele estiver conseguindo fazer a atualização seu ícone continuará como o da Figura 10, mas se ele não conseguir fazer a atualização o ícone ficará como o da Figura 11 até que a comunicação com o servidor ser restabelecida e o *host* volte a ficar sendo atualizado normalmente.

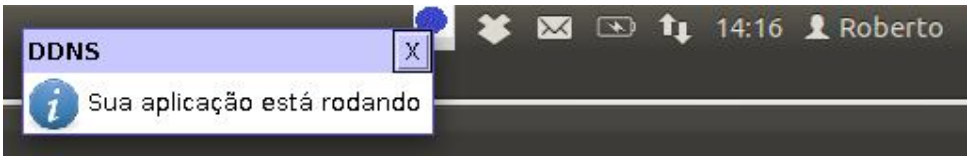

Figura 10 - Atualizando com Sucesso

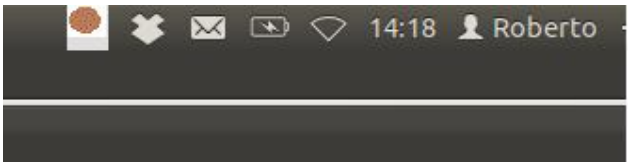

Figura 11 - Problema na Atualização

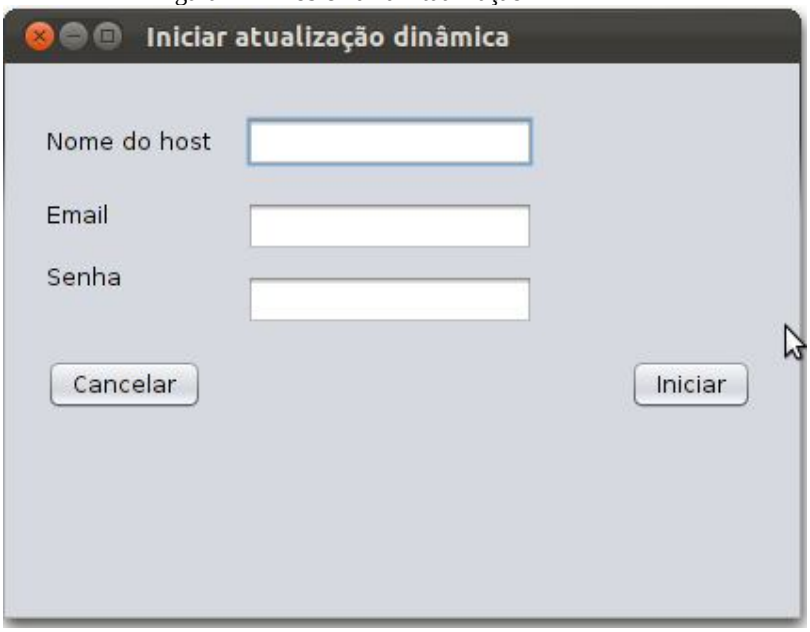

Figura 12 - Iniciar Atualização Dinâmica

5 VALIDAÇÃO

Em nossa validação usamos três máquinas diferentes conectadas em rede. A primeira funcionava como nosso servidor intermediário e servidor DNS, nela tínhamos o servidor *Web*, usamos o *Apache*, o servidor DNS e o banco de dados, usamos o *postgresql*, instalados. A segunda funcionava como nosso cliente, tinha apenas o programa cliente instalado e a mesma requisitava atualização dinâmica de um determinado *host*. A terceira era a máquina que consultava o servidor DNS para saber qual era o endereço IP do *host* da máquina cliente.

Primeiro a máquina cliente colocou um *host* para ser atualizado dinamicamente, e a terceira máquina requereu ao servidor DNS o endereço IP desse host. Depois disso começamos a alterar o endereço IP do *host* e continuamos fazendo consultas ao servidor intermediário para saber com qual endereço IP o *host* estava. Alteramos esse endereço IP dez vezes e em todas as consultas subsequentes de cada mudança o servidor DNS já nos informava o novo endereço IP do *host*. Esses testes foram feitos em uma rede interna e dependendo da hierarquia de servidores DNS essa atualização pode demorar no máximo 5 minutos para se propagar até os clientes que tentam acesa. Para o próprio servidor do serviço a atualização se mostrou instantânea.

6 CONSIDERAÇÕES FINAIS

Ao final desse trabalho conseguimos realizar nosso objetivo principal que era disponibilizar para a sociedade um software de DNS dinâmico de propósito geral, isto é, que também faça DNS reverso para empresas que registram máquinas na própria faixa IP, nosso sistema está disponível em http://dns-dinamico-ufc-quixada.googlecode.com/svn/trunk/, e que esse software fosse livre e de código aberto.

Durante o desenvolvimento desse trabalho, enfrentamos diversas dificuldades, dentre elas tivemos um problema na utilização de TDD, pois essa técnica estava deixando o desenvolvimento muito lento o que iria implicar no não comprimento do cronograma e por conta disso achamos melhor continuarmos o desenvolvimento sem o uso da mesma. Também tivemos bastantes problemas na hora de fazer a atualização de um *host.* A principio, iríamos seguir a RFC 2136 para fazer essa atualização, mas também percebemos que não conseguimos implementá-la, assim fizemos essa atualização reescrevendo o arquivo de

configuração do DNS e reiniciando o serviço. A reinicialização do servidor foi outra atividade que nos consumiu muito tempo, pois não estávamos conseguindo reiniciar o serviço de dentro de nossa CGI, fizemos um script de propriedade do usuário *root* (administrador) e ativamos o bit SUID, mas mesmo assim não estávamos conseguindo executar essa reinicialização. Posteriormente, descobrimos que por questões de segurança o Linux não permite que um script com o bit SUID ativo e de propriedade do root seja executado como tal e portanto fizemos um programa em C para fazer essa reinicialização.

Outra dificuldade encontrada foi na implantação, pois queríamos ter implantado o sistema no servidor da universidade, mas como ainda estamos em período de mudança na infraestrutura do prédio da UFC Quixadá não tivemos acesso ao mesmo.

Para trabalhos futuros sugerimos que seja feita a integração desse sistema com o serviço fone@RNP para viabilizar que esse serviço seja implantado em nosso campus e que seja feito o acompanhamento da macromobilidade desses usuários já que os dados gerados por nosso sistema possibilitam esse acompanhamento. Sugerimos também que seja implementada a RFC 2136 para a atualização dinâmica de nomes de domínio.

REFERÊNCIAS

BERNERS-LEE; FIELDING, R; FRYSTYK, H; **Hypertext Transfer Protocol - HTTP/1.0**, RFC 1945, Maio de 1996.

CALLADO, A. **Interligação e mobilidade de Usuários VoIP no Ceará**. Edital FUNCAP No. 02/2010: Bolsa de Produtividade em Pesquisa e Estímulo à Interiorização, 2010.

COULOURIS, G; DOLLIMORE, J; KINDBERG, T. **Sistemas Distribuídos Conceitos e Projetos.** Tradução João Tortello. Porto Alegre: Bookman, 2007.

DEERING, S.; HINDEN, R.; **Internet Protocol, Version 6 (IPv6) Specification**, RFC 2460, Dezembro de 1988.

DROMS, R.; **Dynamic Host Configuration Protocol**, RFC 2131, Março de 1997.

FIELDING, R; GETTYS, J; MOGUL, J; FRYSTYK, H; MASINTER, L; LEACH, P; BERNERS-LEE; **Hypertext Transfer Protocol -- HTTP/1.1**, RFC 2616, Junho de 1999.

fone@RNP. [site corporativo] **Projeto fone@RNP**. Disponível em: < http://www.rnp.br/voip > Acesso em: 1 de Outubro de 2011.

HINDEN, R.; DEERING, S.; **IP Version 6 Addressing Architecture**. RFC 4291, Fevereiro de 2006.

IBOPE. **Acesso à internet.** Disponível em: < http:/twixar.com/dfjlqGV91 > Acesso em 5 de Outubro de 2011.

KUROSE, J. F.; ROSS, K. W. **Redes de Computadores e a Internet: uma abordagem top-down.** Tradução Opportunity translations. São Paulo: Addison Wesley, 2010.

MOCKAPETRIS, P.; **Domain Names - Concepts and Facilities**, RFC 1034, Novembro de 1987.

MOCKAPETRIS, P.; **Domain Names - Implementation and Specification**, RFC 1035, Novembro de 1987.

MORIMOTO, C. E. **Servidores Linux, guia prático.** Porto Alegre: Sul Editores, 2009.

NO-IP. **Acesso à internet.** Disponível em: < http://www.no-ip.com > Acesso em 03 de Junho de 2012.

POSTEL, J.; **Internet Protocol: DARPA Internet Program Protocol Specification**, RFC 791, Setembro de 1981.

SUPERDNS. **Acesso à internet.** Disponível em: < http://www.superdns.com.br/index.shtml> Acesso em 03 de Junho de 2012.

The Internet Corporation for Assingned Names and Numbers, [site corporativo]. Disponível em: < http://www.icann.org/ > Acesso em 29 de Outubro de 2011

VIXIE, P.; THOMSON, S.; REKHTER, Y.; BOUND, J.; **Dynamic Updates in the Domain Name System (DNS UPDATE)**, RFC 2136, Abril de 1997.

VoIP4all. **Projeto VoIP4all**, [site corporativo]. Disponível em: < http://voip4all.rnp.br > Acesso em: 10 de Outubro de 2011.

ZHAOHUI, L; JIANING, L; Design and implementation of Dynamic Domain Name System Based on BIND. **Computer Network and Multimedia Technology:** International Symposium, Wuhan, p. 1 – 4, Janeiro de 2009.

APÊNDICE

APÊNDICE A – arquivo de configuração do banco de dados

```
CREATE SEQUENCE tabela_tipo_usuario_id_tabela_tipo_usuario_seq;

CREATE TABLE tabela_tipo_usuario (
        id_tipo_usuario INTEGER NOT NULL DEFAULT
nextval('tabela_tipo_usuario_id_tabela_tipo_usuario_seq'),
        nome VARCHAR(15) UNIQUE NOT NULL,
        CONSTRAINT tabela_tipo_usuario_pk PRIMARY KEY (id_tipo_usuario)
);

CREATE SEQUENCE usuario_id_usuario_seq;

CREATE TABLE usuario (
        id_usuario INTEGER NOT NULL DEFAULT nextval('usuario_id_usuario_seq'),
        email VARCHAR(50) UNIQUE NOT NULL,
        Nome_Usuario VARCHAR(20) NOT NULL,
        senha VARCHAR(32) NOT NULL,
        id_tipo_usuario INTEGER NOT NULL,
        CONSTRAINT usuario_pk PRIMARY KEY (id_usuario)
);

CREATE SEQUENCE host_id_seq;

CREATE TABLE tabela_host (
        id_host INTEGER NOT NULL DEFAULT nextval('host_id_seq'),
        id_usuario INTEGER NOT NULL,
        nome_host VARCHAR(50) UNIQUE NOT NULL,
        data_hora TIMESTAMP NOT NULL,
        nat INTEGER NOT NULL,
        ip VARCHAR(40) NOT NULL,
            tipo_ip INTEGER NOT NULL,
        CONSTRAINT tabela_host_pk PRIMARY KEY (id_host, id_usuario)
);

CREATE SEQUENCE atalizacao_id_seq;

CREATE TABLE tabela_atualizacao (
        id_atualizacao INTEGER NOT NULL DEFAULT nextval('atalizacao_id_seq'),
        id_usuario INTEGER NOT NULL,
        data_hora TIMESTAMP NOT NULL,
        ipAtual VARCHAR(40) NOT NULL,
        ipAntigo VARCHAR(40) NOT NULL,
        id_host INTEGER NOT NULL,
        CONSTRAINT tabela_atualizacao_pk PRIMARY KEY (id_atualizacao,
id_usuario)
);
```

```
ALTER SEQUENCE tabela_tipo_usuario_id_tabela_tipo_usuario_seq OWNED BY
tabela_tipo_usuario.id_tipo_usuario;

ALTER SEQUENCE usuario_id_usuario_seq OWNED BY usuario.id_usuario;

ALTER TABLE usuario ADD CONSTRAINT tabela_tipo_usuario_usuario_fk
FOREIGN KEY (id_tipo_usuario)
REFERENCES tabela_tipo_usuario (id_tipo_usuario)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;

ALTER TABLE tabela_host ADD CONSTRAINT usuario_tabela_host_fk
FOREIGN KEY (id_usuario)
REFERENCES usuario (id_usuario)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;

ALTER TABLE tabela_atualizacao ADD CONSTRAINT usuario_tabela_atualizacao_fk
FOREIGN KEY (id_usuario)
REFERENCES usuario (id_usuario)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;

ALTER TABLE tabela_atualizacao ADD CONSTRAINT tabela_host_tabela_atualizacao_fk
FOREIGN KEY (id_host, id_usuario)
REFERENCES tabela_host (id_host, id_usuario)
ON DELETE NO ACTION
ON UPDATE NO ACTION
NOT DEFERRABLE;
```