UNIVERSIDADE FEDERAL FLUMINENSE
CENTRO TECNOLÓGICO
MESTRADO EM ENGENHARIA DE TELECOMUNICAÇÕES

ANTONIO CAMINADA

PULL: UM NOVO MODELO PARA O TRANSPORTE DE CORREIO ELETRÔNICO

Niterói
2007

ANTONIO CAMINADA

PULL: UM NOVO MODELO PARA O TRANSPORTE DE CORREIO ELETRÔNICO

Dissertação apresentada ao Curso de Pós-Graduação em Engenharia de Telecomunicações da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre. Área de Concentração: Processamento de Sinais e Comunicação de Dados Multimídia.

Orientador: Prof. Dr. LUIZ CLÁUDIO SCHARA MAGALHÃES

Niterói
2007

ANTONIO CAMINADA

PULL: UM NOVO MODELO PARA O TRANSPORTE DE CORREIO ELETRÔNICO

Dissertação apresentada ao Curso de Pós-Graduação em Engenharia de Telecomunicações da Universidade Federal Fluminense, como requisito parcial para obtenção do Grau de Mestre. Área de Concentração: Processamento de Sinais e Comunicação de Dados Multimídia.

Aprovada em ____ de _____ de _____

BANCA EXAMINADORA

_____

Prof. Luiz Cláudio Schara Magalhães, D.Sc – Orientador
Universidade Federal Fluminense

_____

Prof Michael Stanton, D.Sc.
Universidade Federal Fluminense

_____

Prof. Alexandre Grosjgold, D.Sc.
Diretor de Operações da Rede Nacional de Ensino e Pesquisa
Docteur Ingenieur, Paris VI

Niterói
2007

Dedico esta dissertação à minha família, meus pais e meu irmão.
Dedico também aos amigos da Dynamis, cujo apoio foi fundamental para o sucesso deste trabalho.
Finalmente à Finha, meu amor e meu ponto de equilíbrio.

AGRADECIMENTOS

Ao meu orientador, Prof. Luiz Cláudio Schara Magalhães, que apostou em mim, me dando a oportunidade de fazer este mestrado e pela orientação sempre presente e construtiva.

Ao meu irmão Nuno, meus amigos Albert, Randolpho e Mônica, que me guiaram no difícil mundo do JAVA, me ajudando a construir o protótipo de testes.

Aos amigos da Dynamis – Adelso, Oliver, Sérgio e Cláudio - pelo tempo cedido para que eu pudesse assistir às aulas e estudar.

A todos os professores do curso de mestrado, por me abrirem as portas para um mundo novo.

Ao amigo Argollo, que no pouco tempo em que nos conhecemos, fez uma grande contribuição a este trabalho.

Aos colegas de curso, pela ajuda e companheirismo

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

O correio eletrônico é uma das aplicações de maior sucesso da *Internet*. Por meio dele é possível enviar, de forma quase instantânea e a um custo muito baixo, mensagens compostas apenas de texto ou que contenham arquivos anexados (vídeos, fotografias, documentos, etc.). No entanto, algumas características de sua arquitetura atual, assim como o uso mal-intencionado, fizeram com que alguns problemas surgissem, afetando a usabilidade e ameaçando o futuro da aplicação. Esta dissertação apresenta um novo modelo de transporte de correio eletrônico que divide a transferência da mensagem em duas partes: uma mensagem-resumo é enviada ao destinatário usando o SMTP, e a mensagem original é armazenada em um novo servidor, proposto nesta dissertação, e pode ser acessada por vontade explícita do destinatário, via HTTP. Este modelo tem várias vantagens sobre o modelo usado atualmente, a principal sendo requerer um servidor com presença constante na Internet para permitir a busca das mensagens, o que pode inibir o *spam*.

Palavras chave: Correio Eletrônico, SMTP, HTTP, MIME, transporte.

ABSTRACT

Electronic mail is one of the most successful Internet applications. With it it is possible to send messages almost instantaneously and at a very low cost. These messages might be text-only or they can have attachments (videos, still images, spreadsheets, etc.). But some of its architectural characteristics combined with malicious use have created problems, which affect its usability and its future. This thesis presents a new model for sending electronic mail, which breaks the transfer in two parts: the e-mail header is sent using SMTP, but the body is kept in a new server, defined in this thesis, and can be downloaded via HTTP. This model has several advantages over the current model, the main advantage being the requirement of a server that has to be connected full time to the Internet to allow the recipients of e-mail to have access to it, which may inhibit spam.

Keywords: e-mail, SMTP, HTTP, MIME, transport.

1. INTRODUÇÃO

O correio eletrônico (ou *e-mail*) é uma aplicação de redes que permite o envio de mensagens entre usuários da *Internet* ou de redes corporativas. O sucesso dessa aplicação é tão grande que ela se tornou uma ferramenta indispensável, seja no trabalho ou no lazer, por sua facilidade e versatilidade de uso. Com ela podemos enviar sons, imagens (fotos e vídeos) e praticamente qualquer tipo de arquivo eletrônico anexado a um custo muito baixo e quase instantaneamente. Com o passar do tempo, os usuários descobriram novas utilidades para o correio eletrônico que extrapolam sua finalidade original, que era simplesmente enviar arquivos texto. Hoje ele é utilizado como agenda (pessoas mandam lembretes para elas mesmas sobre compromissos importantes), como um meio de transferir arquivos (em substituição ao FTP[1]), repositório remoto de dados (um arquivo pode ser acessado de qualquer parte do mundo onde haja uma conexão de *Internet*) e até como espaço de armazenagem e *backup* (alguns provedores oferecem grandes espaços de armazenamento de mensagens - acima de 1 *gigabyte* – que são utilizados para guardar cópias de segurança de arquivos importantes).

A grande aceitação do correio eletrônico pode ser explicada por vários fatores, se comparado a outros meios de comunicação. Algumas vantagens de uma carta eletrônica sobre

[1] FTP – File Transfer Protocol – protocolo de transferência de arquivos, definido na RFC 959 [58]

outras formas de comunicação como uma chamada telefônica, programas de mensagens instantâneas (*instant messengers*), *facsimile* (fax) e o correio tradicional são:

- A composição da carta é feita antes do envio, possibilitando que sejam feitos vários rascunhos antes de se chegar à mensagem final, aumentando a clareza e inteligibilidade. Já no telefone e programas de mensagens instantâneas, a conversa acontece em tempo real. Uma palavra dita sem intenção pode causar um grande mal-entendido.

- Não requerer sincronização entre os participantes da comunicação, ou seja, se um dos participantes não estiver disponível naquele momento, a mensagem pode ser enviada assim mesmo, podendo ser lida depois, ao contrário do telefone e dos programas de mensagens instantâneas, que requerem a presença de ambos os participantes simultaneamente.

- A carta, estando em meio eletrônico, poder ser manipulada pelo destinatário, ao contrário do facsimile, em que a mensagem está impressa em papel.

- O correio eletrônico é muito mais rápido que o correio normal [1].

- Permite o envio de arquivos anexados.

- Baixo custo. Uma conta de correio eletrônico, hoje, tem custo zero, pois muitos provedores oferecem o serviço gratuitamente.

Apesar da evolução em termos de funcionalidades (originalmente o correio eletrônico suportava apenas mensagens de texto; hoje existe a possibilidade de se enviar arquivos de diversos formatos anexados à mensagem), sua arquitetura permaneceu relativamente inalterada (o modelo *Push*, por exemplo, – que encaminha todas as mensagens para o destinatário independentemente de sua vontade de recebê-las – era uma necessidade nos primórdios da *Internet*, quando esta utilizava redes do tipo *store-and-foward*, que faziam com que fosse necessário encaminhar as mensagens para o próximo nó quando este estivesse conectado, hoje se tornou desnecessário, sendo o correio eletrônico uma das poucas aplicações que ainda o utiliza – ver seção 2.1). Problemas de segurança (as mensagens são enviadas em texto sem criptografia, não existe autenticação do remetente, entre outros – ver

seção 2.5) e do volume de tráfego na *Internet* se somam a outros (envio de *spam*, vírus, etc) que, se não resolvidos, podem inviabilizar o correio eletrônico no futuro.

O problema que se apresenta hoje, e que é assunto desta dissertação, é como corrigir ou atenuar essas falhas do modelo atual sem inviabilizar o envio de mensagens. Para haver uma mudança no protocolo (uma nova versão do SMTP[2] ou um protocolo inteiramente novo), seria necessário um período de transição, onde o novo e o antigo modelo pudessem coexistir. Isso seria complicado, pois exigiria coordenação entre milhares de provedores de acesso à *Internet*[3] ao redor do mundo e não há garantias de que máquinas usando o novo protocolo se comunicariam adequadamente com máquinas ainda utilizando o antigo protocolo.

Portanto, é necessário que qualquer mudança no correio eletrônico garanta a operabilidade do sistema como um todo, e a maneira mais fácil de fazer isso é manter o SMTP inalterado, apenas mudando sua forma de utilização, como foi feito quando se adicionou ao correio eletrônico a capacidade de enviar arquivos multimídia através do MIME[4] (ver seção 2.4). Por isso esta proposta de mudar o transporte do correio eletrônico, substituindo o modelo *Push* por um modelo *Pull*. Neste modelo o destinatário receberá uma pequena notificação sobre a existência de mensagens para ele, que estarão armazenadas no servidor do remetente. Ele poderá, então, escolher que mensagens deseja ler e receber apenas estas. Essa mudança, apesar de originalmente ter sido pensada para resolver apenas o problema enfrentado por usuários que se conectam à *Internet* por enlaces com baixas taxas de transferência e que perdem muito tempo recebendo mensagens indesejadas (ver seção 1.2), minimiza outros problemas (diminui o risco de infecção por vírus, diminui o volume de tráfego na *Internet*, dificulta o envio de *spam*, etc.) e pode ser implementada sem problemas, pois não há incompatibilidade entre os modelos novo e antigo.

1.2 MOTIVAÇÃO DESTA DISSERTAÇÃO

O correio eletrônico, apesar de ser uma das aplicações de maior sucesso da *Internet*, tem problemas de arquitetura que podem ser explorados por todo tipo de pessoas mal-

[2] SMTP – Simple Mail Transfer Protocol – definido da RFC 2821 [2].
[3] ISP – Internet Service Provider – provedores de acesso à Internet, que oferecem contas de correio eletrônico
[4] MIME – Multipurpose Internet Mail Extensions – definido nas RFC 2045 [6], 2046 [36], 2047 [37], 2048 [38] e 2049 [39].

intencionadas, e tem também, problemas inerentes à maneira como foi inicialmente concebido e implementado.

O presente trabalho, a princípio, tinha por objetivo resolver dois desses problemas:

- **O tempo de transferência das mensagens (muitas indesejadas) do servidor POP para o UA do destinatário:** em enlaces com baixa taxa de transferência, perde-se muito tempo para fazer a transferência das mensagens armazenadas no servidor POP que não serão nem mesmo lidas, por se tratarem de *spam*. O destinatário é penalizado porque tem que manter uma conexão aberta (muitas vezes por acesso discado) por longos períodos para receber, no final, mensagens que ele simplesmente apaga sem ler.

  Como atualmente muitos usuários já utilizam acessos de banda larga (cabo, DSL[5], etc.) em casa, este problema está sendo mais evidente na telefonia celular, onde a cobrança é feita por Kb transmitido.

  Com a implementação proposta, o tempo (e o custo) de *download* será consideravelmente reduzido, pois a mensagem-resumo enviada é muito pequena (nos testes realizados seu tamanho aproximado foi de 2Kb) e pode ser recuperada rapidamente, mesmo em enlaces com baixas taxas de transmissão.

- **A utilização do espaço de armazenagem do destinatário em sua caixa postal (caixa de entrada):** como as mensagens recebidas são armazenadas na caixa postal do destinatário, com freqüência ela fica cheia, impedindo então, que o usuário receba novas mensagens. Isso obriga os usuários a fazer *downloads* freqüentes para liberar espaço na caixa postal, o que pode não ser possível ou pode ser inconveniente, como por exemplo, quando ele sai de férias.

  No modelo *Pull* as mensagens ficam armazenadas no servidor do remetente, resolvendo o problema de armazenagem e também resolvendo um problema conceitual, que é a transferência do ônus da comunicação para quem a inicia, como acontece com o correio tradicional, as ligações telefônicas, etc.

[5] *Digital Subscriber Line* – é uma tecnologia que permite a transmissão digital de dados por linhas telefônicas

Com o passar do tempo, constatamos que o novo modelo de transporte (*pull*) poderia resolver (ou minimizar) outros problemas, como *spam*, vírus e *worms*, volume do tráfego na *Internet* e falta de segurança, entre outros.

1.3 OBJETIVO

O objetivo dessa dissertação é apresentar um novo modelo de transporte de correio eletrônico (modelo *Pull*) no qual, ao contrário do modelo atual (*Push*), as mensagens só sejam vistas pelo destinatário se ele expressamente as requisitar. Isso será feito através da criação de uma mensagem-resumo, que conterá informações suficientes para que o destinatário avalie se quer ou não ver a mensagem original, e um link HTTP[6] para a mensagem, que será exibida no navegador (*browser)* padrão da máquina do destinatário. A criação dessa mensagem-resumo se dará no servidor SMTP (pelo programa descrito nessa dissertação) do remetente, onde também ficará armazenada a mensagem original, transferindo o ônus da comunicação para quem a inicia, como ocorre com outras formas de comunicação, como o telefone e o correio tradicional, por exemplo.

A dissertação está estruturada da seguinte forma: no capítulo 2 será descrito como funciona o correio eletrônico hoje, sua arquitetura, os problemas que mais afetam a aplicação e as soluções disponíveis para os problemas que ele enfrenta. No capítulo 3 será descrita a arquitetura proposta, discorrendo também sobre suas vantagens e limitações. No capítulo 4 será mostrado como foi feita a prova do conceito, descrevendo como funciona o protótipo implementado e mostrando como este trabalho será complementado. No capítulo 5 será feita uma discussão sobre o futuro do correio eletrônico, indicando quais as tendências que hoje se mostram mais prováveis e as conclusões obtidas.

[6] HTTP - Hypertext Transfer Protocol – definido na RFC 2616 [59].

2. O CORREIO ELETRÔNICO

O correio eletrônico[7] é um conjunto de protocolos e aplicações que permite que mensagens de texto sejam enviadas pela *Internet* (ou através de uma rede local). Mesmo quando uma mensagem contém imagens, sons ou arquivos anexados, estes são convertidos em texto (texto aqui significando um conjunto de caracteres reconhecidos e não algo legível) através do MIME (*Multipurpose Internet Mail Extensions* [6, 36, 37, 38, 39]).

Analogamente ao correio tradicional, uma mensagem de correio eletrônico tem duas partes: a mensagem propriamente dita e um invólucro (cabeçalho), que contém informações de controle para entrega (por exemplo, o endereço do remetente e do destinatário) como o envelope de uma carta tradicional. Assim, a mensagem é formada por um cabeçalho com uma estrutura bem definida e um corpo (texto), que pode ser estruturado (usando MIME). O formato de uma mensagem de correio eletrônico pode ser encontrado na RFC 2822 [32]. Na figura 1 é exibido um exemplo de mensagem de correio eletrônico, sem arquivo anexado:

[7] O termo correio eletrônico é usado, indistintamente, tanto para a mensagem propriamente dita quanto para a ferramenta responsável pelo envio da mensagem.

Return-Path: <remetente@servidor.com.br>
Received: from smtp.remetente.com.br (200.221.4.208) by calipso.destinatario.com (7.0.016.2)
id 452CFA6F0003F5A8 for destinatario@servidor.com.br; Sun, 15 Oct 2006 20:01:29 -0200
Received: from localhost (localhost [127.0.0.1])
by socom7.servidor.com.br (Postfix) with ESMTP id 7D2EA112
for <destinatario@servidor.com.br>; Sun, 15 Oct 2006 19:01:29 -0300 (BRT)
Received: from remetente (unknown [201.19.67.109])
by socom7.servidor.com.br (Postfix) with SMTP id 3DEBD96E
for <destinatario@servidor.com.br>; Sun, 15 Oct 2006 19:01:29 -0300 (BRT)
Message-ID: <016901c6f0a5$7a882290$6400a8c0@remetente>
Reply-To: "Remetente" <remetente@servidor.uff.br>
From: "Remetente" <remetente@servidor.com.br>
To: "Destinatário" <destinatario@servidor.com.br>
References: <001301c6f0a4$c6dc7fc0$7b00a8c0@antb2ad3c08645>
Subject: =?iso-8859-1?Q?Re:_Vers=E3o_18?=
Date: Sun, 15 Oct 2006 20:01:32 -0200
X-Priority: 3
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express 6.00.2900.2869
X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2900.2962
X-SIG5: 1d696675da12450f322f3cf9a7542a74
X-Antivirus: AVG for E-mail 7.1.408 [268.13.9/490]
Mime-Version: 1.0
Content-Transfer-Encoding: 8bit
Content-Type: text/plain; format=flowed; charset=iso-8859-1; reply-type=original

Texto da mensagem.

--remetente

----- Original Message -----
From: "Remetente" <remetente@servidor.com.br>
To: <destinatario@servidor.uff.br>
Sent: Sunday, October 15, 2006 7:56 PM
Subject: Versão 18

> Texto da mensagem

Figura 1: mensagem de correio eletrônico

A parte em negrito é o cabeçalho da mensagem e o restante, o corpo.

2.1- HISTÓRICO

O modelo de correio eletrônico atual surgiu em 1971 pelas mãos de Ray Tomlinson [7], que na época trabalhava em um programa que permitiria que usuários compartilhando um mesmo computador deixassem mensagens uns para os outros. Ao mesmo tempo ele também desenvolvia um programa para transferência de arquivos entre máquinas ligadas à ARPANET[8]. Tomlinson percebeu que, se juntasse os dois programas, poderia enviar mensagens através da rede.

Um de seus problemas era como distinguir as mensagens que eram para a máquina em uso e para fora (rede). Ele teve a idéia então, de utilizar o símbolo @ (*at*). Segundo Tomlinson: "Ele designa um lugar e é a única preposição do teclado" [7].

O correio eletrônico da *Internet* também descende dos programas de BBS, como a FIDONET [8, 9,10], de programas de envio de arquivos como o UUCP [11] e do correio eletrônico da BITNET [12]. Destes, ele herdou o modelo de *Push*. Tanto a FIDONET quanto a BITNET eram redes do tipo *store-and-forward*[9]. Enquanto na BITNET o endereço era plano (usuário@sistema), na FIDONET o endereço era hierárquico (como na *Internet*, mas com zona:rede/nó.ponto), e no UUCP o endereço continha as informações de roteamento explícito da mensagem, sendo uma seqüência de nomes de máquina separados por pontos de exclamação (*source-routing*). A conectividade dos sistemas de BBS e do UUCP era esporádica. Para a carta chegar ao destino, ela passava por vários sistemas intermediários, que poderiam ou não estar *online* (ativos). A entrega das mensagens dependia dos sistemas intermediários entrarem em contato, na maior parte das vezes via linhas discadas, por isso, fazia sentido levar a mensagem para mais perto do usuário (usando o modelo *Push*) - a cada contato, mensagens fluíam de um sistema para o outro. Esse contato podia ocorrer a cada hora ou até mesmo uma vez por dia (ou menos), dependendo do custo (ligações de longa distância, como conexões internacionais, ocorriam com baixa freqüência). Apesar da *Internet* não requerer o modelo *Push*, já que os sistemas normalmente ficam constantemente conectados, era bem entendido que a utilidade do correio eletrônico aumentava conforme fosse maior o número de pessoas conectadas a ele. Assim, era importante que o correio da *Internet* fosse

[8] Advanced Research Projects Agency Network – rede do departamento de defesa dos EUA, criada nos anos 70, que tinha por objetivo conectar universidades e departamentos de pesquisa do governo norte-americano.

[9] Tipo de rede em que a mensagem é armazenada em um nó intermediário antes de ser encaminhada para o próximo nó, que pode ser outro nó intermediário ou o destino final.

compatível com o modelo existente até então (foram criados vários *gateways*[10] para permitir o fluxo de mensagens para sistemas de correio diferentes). Daí o modelo *Push* ter sido adotado, e funcionar até hoje, apesar de ser um dos poucos serviços da *Internet* que segue este modelo (os outros, como *webcasting* [13], não obtiveram o mesmo sucesso e são pouco comuns atualmente). Mais informações sobre a história do correio eletrônico podem ser encontradas em [1].

2.2 A ARQUITETURA ATUAL

Na Figura 2 podemos ver o sistema de transporte para o correio eletrônico na *Internet*.

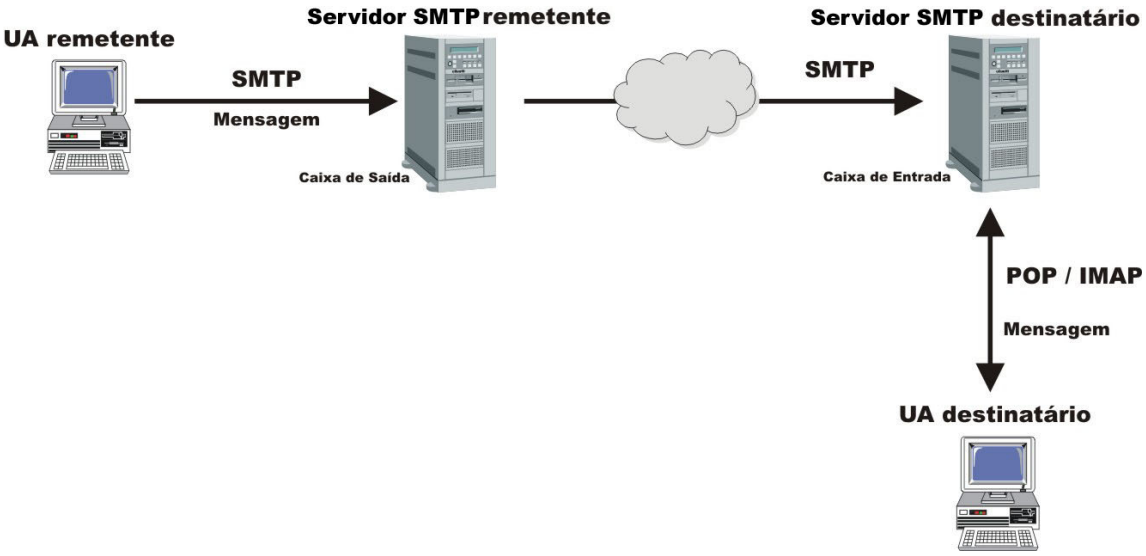

Figura 2: Sistema de Transporte de Correio Eletrônico na *Internet*

Atualmente o correio eletrônico funciona da seguinte forma: o usuário escreve a mensagem em um *user agent* (UA), que é um programa utilizado para compor, enviar e receber mensagens - como por exemplo, o Outlook Express e que isola o usuário dos detalhes de implementação do sistema, como a troca de informações de controle entre servidores, por exemplo. Ao clicar em "enviar", a mensagem é remetida ao servidor SMTP do remetente, onde está sua caixa de saída (local onde ficam as mensagens antes de serem enviadas ao

[10] Gateway: máquina destinada a interligar redes, separar domínios de colisão, ou mesmo traduzir protocolos.

destinatário). O SMTP (*Simple Mail Transfer Protocol*) especificado na RFC 2821 [2] é um protocolo assimétrico[11], encarregado de encaminhar as mensagens de correio eletrônico do UA do remetente ao servidor SMTP do destinatário (onde fica sua caixa de entrada – local onde as mensagens ficam armazenadas até que o usuário as apague ou as transfira para sua máquina). Ele se baseia no seguinte modelo de comunicação: como resultado de um comando do usuário, o SMTP remetente estabelece um canal bi-direcional com o SMTP destinatário, que pode ser o destino final ou um sistema intermediário. Os comandos SMTP são gerados pelo SMTP transmissor e enviados para o SMTP receptor. As respostas são geradas pelo receptor em conseqüência dos comandos. Uma vez estabelecido um canal de comunicação, o transmissor SMTP envia um comando MAIL, indicando o remetente da mensagem. Se o receptor SMTP puder aceitar a mensagem, ele responde com um OK. O transmissor, então, envia um comando RCPT, indicando o destinatário da mensagem. Se o receptor SMTP puder aceitar mensagens para o destinatário, ele responde com um OK. Caso contrário, responde com um comando que rejeita aquele usuário (mas não acaba com a transação). O transmissor e o receptor podem negociar vários destinatários. Quando todos os destinatários tiverem sido negociados, o transmissor envia um comando DATA e o receptor trata tudo após esse comando como o corpo da mensagem, que termina com uma linha cujo único caractere visível é um sinal de ponto (".") (existem, nesta linha, os caracteres invisíveis CRLF[12]). Durante o caminho entre o remetente e o destinatário final a mensagem pode passar por vários MTAs - *Mail Transfer Agents* – agentes intermediários que fazem uma "ponte" entre o remetente e o destinatário. Na Figura 3 é exibido um exemplo de uma transação SMTP típica [2]:

[11] Protocolo assimétrico é um protocolo que faz a comunicação entre máquinas clientes e servidores, ao contrário de protocolos simétricos, que fazem a comunicação entre máquinas que não são clientes ou servidores entre si.

[12] CRLF – Carriage Return Line Feed – uma referência às antigas máquinas de escrever onde, ao final de uma linha, o carro era empurrado para a esquerda e uma nova linha em branco era mostrada.

S: 220 foo.com Simple Mail Transfer Service Ready

C: EHLO bar.com

S: 250-foo.com greets bar.com

S: 250-8BITMIME

S: 250-SIZE

S: 250-DSN

S: 250 HELP

C: MAIL FROM:<Smith@bar.com>

S: 250 OK

C: RCPT TO:<Jones@foo.com>

S: 250 OK

C: RCPT TO:<Green@foo.com>

S: 550 No such user here

C: RCPT TO:<Brown@foo.com>

S: 250 OK

C: DATA

S: 354 Start mail input; end with <CRLF>.<CRLF>

C: Blah blah blah...

C: ...etc. etc. etc.

C: .

S: 250 OK

C: QUIT

S: 221 foo.com Service closing transmission channel

Figura 3: seqüência de mensagens de controle entre cliente e servidor SMTP

Onde C representa o cliente SMTP e S o servidor SMTP.

Para sistemas que não estejam permanentemente conectados à *Internet*, foram desenvolvidos ainda outros protocolos: POP (*Post Office Protocol*), atualmente na versão 3 [14] e IMAP [15] (*Internet Message Access Protocol*). Estes protocolos são encarregados de transferir (*download*) as mensagens do servidor SMTP do destinatário para seu UA. A grande diferença entre o POP e o IMAP é a localização das mensagens. No IMAP elas ficam no servidor, sendo apenas lidas no UA. No POP elas podem ser copiadas para o cliente (e,

normalmente, apagadas do servidor), o que torna a leitura a partir de múltiplos computadores menos confortável.

Devido ao sistema de transporte ter sido fixado em ASCII de 7 bits[13] na RFC 821[14] [32] as mensagens são enviadas como texto, mesmo quando tiverem um formato interno complexo. Por exemplo, se contiverem arquivos (sons, imagens, vídeos, planilhas, etc) ou incluirem diversas formas alternativas para serem exibidas conforme as capacidades do UA do destinatário. Isso é possível graças ao MIME [6] – *Multipurpose Internet Mail Extensions.* O MIME não só define codificações para a transformação de conteúdo não ASCII para ASCII, bem como um cabeçalho de definição do formato da mensagem (ver seção 2.4).

No destinatário final o texto é convertido novamente para o formato original pelo UA.

2.3 PRINCIPAIS RFCS RELACIONADAS AO CORREIO ELETRÔNICO

O correio eletrônico não está definido em uma única RFC[15]·, mas em várias RFCs relacionadas (em ordem cronológica):

- As RFCs 1122 [43] e 1123 [44] definem os requisitos para a implementação do conjunto de protocolos da *Internet* (*Internet Protocol Suite*), em máquinas (*hosts*) ligadas à *Internet*. Estas RFCs de 1989 enumeram os protocolos que um *host* ligado à *Internet* deve suportar e também, para cada protocolo, um conjunto de características necessárias, recomendadas e opcionais.

- As RFCs 1421 [45], 1422 [46], 1423 [47] e 1424 [48] definem procedimentos (criptografia, autenticação, algoritmos, etc.) para aumentar a privacidade das mensagens durante o transporte. Apresentado em 1993, este conjunto de RFCs define procedimentos para a autenticação e criptografia (compatível com chaves simétricas e assimétricas) de mensagens para prover PEM (*Privacy-Enhanced Mail*). Este conjunto de serviços (confidencialidade, autenticação, verificação da integridade e certeza da origem da mensagem) é obtido através do uso de criptografia fim-a-fim, não sendo necessário alterar os mecanismos de transporte intermediários.

---

[13] O SMTP já foi estendido para permitir o trânsito de informação com 8 bits [2]

[14] A RFC 821 foi atualizada pela RFC 2821

[15] Request For Comments – documentos que descrevem padrões adotados na *Internet*

- RFC 1855 [38] No começo da *Internet*, os usuários eram pessoas com conhecimentos técnicos e que acompanharam o desenvolvimento da Rede. Eles sabiam como se comportar e tinham conhecimento das normas de conduta aceitáveis. Com a popularização da *Internet*, muitos usuários leigos passaram a acessar a rede sem o mesmo conhecimento técnico. Esta RFC apresentada em 1995, contém uma série de regras de etiqueta (*Netiquette Guidelines* - como se apresentar, o que é próprio de ser enviado e o que não é, a quem pertencem as mensagens, etc.) a serem usadas na troca de mensagens, para que estes usuários novatos se adeqüem à "cultura da *Internet*".

- RFC 1869 [39] descreve um arcabouço que estende o SMTP, permitindo que um servidor SMTP informe a um cliente SMTP que extensões ele suporta. Apesar de o SMTP ser um protocolo robusto, estável e eficiente, as marcas do tempo se tornam evidentes, fazendo-se necessário estendê-lo para acompanhar as mudanças na tecnologia, nos usos e costumes e no modo como o correio eletrônico é usado hoje. Esta RFC apresentada em 1995 define como servidores e clientes estendidos podem informar um ao outro que extensões suportam.

- RFC 1870 [40] Com o advento do MIME, uma conseqüência foi que o SMTP passou a transportar mensagens de vários tamanhos, em função dos arquivos anexados (antes não suportados). Isto traz problemas novos para os servidores, que terão que se preocupar com fatores como espaço em disco para armazenar mensagens potencialmente grandes. Esta RFC de 1995 descreve uma extensão que possibilita ao servidor SMTP recusar uma mensagem (mesmo que temporariamente) com base em seu tamanho estimado pelo cliente SMTP.

- RFC 1893 [41] Quando esta RFC foi proposta em 1996, não havia um mecanismo padrão para informar erros de sistema no transporte de mensagens, além do conjunto limitado de erros do SMTP. Esta RFC propõe um novo conjunto de códigos para melhorar a notificação de status de entrega ou erro na entrega de mensagens.

- RFC 1939 [14] especifica o POP (*Post Office Protocol*). O POP é um protocolo amplamente utilizado para recuperar (*download*) mensagens de um servidor SMTP. Ele permite que o usuário se conecte, transfira as mensagens para seu *user agent* e encerre a conexão (existe a opção de deixar as mensagens no servidor SMTP), não sendo necessário ter uma conexão permanente de acesso à *Internet*. A RFC 1939 é de

1996 e descreve a versão 3 do POP, que foi apresentado pela primeira vez em 1984 na RFC 918 [66].

- As RFCs 2045 [6], 2046 [33], 2047 [34], 2048 [35] e 2049 [36] descrevem o MIME - *Multipurpose Internet Mail Extensions*. Elas definem, entre outras coisas, como codificar arquivos anexados que não estão em texto ASCII (ver seção 2.4).

- RFC 2060 [15] especifica o IMAP (*Internet Message Access Protocol*). O IMAP permite que usuários acessem e manipulem suas mensagens diretamente no servidor SMTP. Com ele é possível manipular (criar, renomear e apagar) as caixas de correio (*mailboxes*) no servidor SMTP, como se estivessem na máquina local. Além disso, é possível o compartilhamento de caixas postais entre membros de um grupo de trabalho e fazer pesquisas por mensagens usando palavras-chave. Uma das desvantagens do IMAP é que, como as mensagens ficam armazenadas no servidor, o espaço de armazenagem é definido pelo servidor, e não pelo usuário. O IMAP foi originalmente apresentado na RFC 1064 [67], de 1988, e a versão atual data de 1996.

- RFC 2487 [42] descreve uma extensão ao serviço SMTP que possibilita a um cliente e servidor SMTP usarem TLS (*Transport-Layer Security*) [58] para que possam manter uma comunicação segura através da *Internet*. Clientes e servidores SMTP se comunicam por texto (legível por humanos) através da *Internet*. Freqüentemente, estas mensagens passam por redes que não são conhecidas (e portanto não são confiáveis) dos clientes e servidores em questão. Além disso, muitas vezes seria desejável que clientes e servidores pudessem se autenticar mutuamente para poder estabelecer uma comunicação segura. Esta RFC de 1999 propõe um mecanismo pelo qual o SMTP funcionaria sobre TLS (que foi derivado de SSL - *Security Sockets Layer*), com a possibilidade de autenticação e transmissão segura das mensagens.

- RFC 2646 [37], apresentada em 1999, define um novo parâmetro (*Format*), que pode assumir os valores *Fixed* e *Flowed*, para permitir que parágrafos de texto no formato *text/plain* possam ser corretamente exibidos. Esta necessidade surgiu de problemas de interoperabilidade entre sistemas que não conseguiam exibir corretamente tipos de mídia definidos como *text/enriched* ou *text/html*.

- RFC 2821 [2] define o *Simple Mail Transfer Protocol* ou SMTP. Apresentada em abril de 2001, esta RFC atualiza e amplia a RFC 821 [63], que é a especificação original do SMTP. Estas duas RFCs definem como o correio eletrônico é enviado pela *Internet* especificando, entre outras coisas, o modelo de transporte e a sintaxe do protocolo. A RFC 2821 difere da 821 por deixar de lado algumas características do modelo original que não estavam em uso massificado na *Internet* em meados dos anos 90. Além de aposentar as RFCs 821 e 974 [64] ela atualiza o modelo de transporte proposto na RFC 1123 [44].

- RFC 2822 [32] define o formato básico das mensagens de correio eletrônico, como por exemplo os cabeçalhos (To:, Cc:, Subject:, etc) e o corpo (usando texto padrão ASCII). Este padrão é para mensagens no formato texto apenas, não fazendo nenhuma menção à transmissão de imagens, áudio ou qualquer outro tipo de dados estruturados. Apresentada em abril de 2001, ela amplia e aposenta a RFC 822 [65], atualizando-a para refletir práticas atuais e incorporar mudanças introduzidas por outras RFCs.

2.4 O MIME - *MULTIPURPOSE INTERNET MAIL EXTENSIONS*

O MIME é um conjunto de regras que estende o modelo original de correio eletrônico, possibilitando o suporte a:

- Textos em conjuntos de caracteres que não sejam US-ASCII;

- Arquivos anexos (sons, imagens, planilhas, etc.);

- Corpos de mensagens com múltiplas partes (por exemplo, vários arquivos independentes entre si anexados a uma mensagem);

- Informações de cabeçalho em conjuntos de caracteres que não sejam US-ASCII;

O MIME define campos do cabeçalho da mensagem que especificam, entre outras coisas, que àquela é uma mensagem MIME (*mime-version*), o tipo de conteúdo (*content-type*) e, se a mensagem for do tipo *multipart*, a fronteira (*boundary*) entre os tipos de mídia (texto, fotos, sons, etc) e o tipo de codificação usada para converter os arquivos anexados, de binário

para texto (*content-transfer-encoding*). Estes campos indicam a estrutura da mensagem e como ela deve ser exibida.

Uma das características mais importantes oferecidas pelo MIME é a possibilidade de estruturar a mensagens em partes, de acordo com o tipo de mídia. A tabela abaixo mostra os tipos de partes que são definidos pelo subtipo *multipart* do cabeçalho *content-type*:

Subtipo do Multipart	Relação entre as partes
mixed	As partes não são relacionadas entre si, por exemplo, arquivos anexados.
related	As partes são componentes de um único documento, por exemplo, uma página HTML e as imagens dessa página.
parallel	As partes devem ser exibidas simultaneamente, por exemplo, áudio e vídeo.
alternative	As partes contêm o mesmo conteúdo em formatos alternativos, por exemplo, uma página HTML e o texto puro equivalente
digest	As partes são mensagens de correio eletrônico

Tabela 2: subtipos do tipo *multipart* [67]

Para separar as partes existe um elemento *boundary*, que indica o início e o fim de uma parte (ver exemplo no final desta seção).

O envio de arquivos não-texto (sons, imagens, etc.) junto com uma mensagem de correio eletrônico é feito convertendo o arquivo binário em texto ASCII usando codificações especificadas no cabeçalho da mensagem, como por exemplo, as codificações BASE64 [16] e *Quoted-Printable*.

Na BASE64 os caracteres usados para a codificação são A-Z, a-z, 0-9, + e /, que tem grande possibilidade de existir em todos os conjuntos de caracteres. Isso permite que, mesmo que o texto codificado seja convertido para outra codificação (ASCII para EBCDIC, por exemplo), o conteúdo ainda possa ser recuperado pois apesar de seu valor numérico variar para cada conjunto de caracteres, o caractere em si não será mudado.

A codificação BASE64 funciona da seguinte forma: cada grupo de 24 bits recebidos é codificado (da esquerda para a direita) concatenando-se três grupos de oito bits. Esses 24 bits são, então, tratados como quatro grupos de seis bits, sendo cada grupo traduzido para um caractere no alfabeto da BASE64 (ver tabela abaixo).

Valor	Codificação	Valor	Codificação	Valor	Codificação	Valor	Codificação
0	A	17	R	34	i	51	z
1	B	18	S	35	j	52	0
2	C	19	T	36	k	53	1
3	D	20	U	37	l	54	2
4	E	21	V	38	m	55	3
5	F	22	W	39	n	56	4
6	G	23	X	40	o	57	5
7	H	24	Y	41	p	58	6
8	I	25	Z	42	q	59	7
9	J	26	a	43	r	60	8
10	K	27	b	44	s	61	9
11	L	28	c	45	t	62	+
12	M	29	d	46	u	63	/
13	N	30	e	47	v		
14	O	31	f	48	w	(pad)	=
15	P	32	g	49	x		
16	Q	33	h	50	y		

Tabela 2: A tabela de tradução para a codificação BASE64

A codificação *Quoted-Printable* é usada para representar mensagens que consistam principalmente de octetos que correspondam a caracteres imprimíveis no conjunto ASCII. Ela permite que caracteres de 8 bits sejam representados usando caracteres ASCII de 7 bits e conseqüentemente, transmitidos por rotas que suportam codificação de 7 bits. Se a maior parte dos caracteres for de 7 bits, é melhor apenas converter os que não são, de 8 bits para 7 bits do que usar a BASE 64, que incha o arquivo em um terço.

Qualquer caractere de oito bits pode ser convertido em três caracteres: o sinal de igual "=" seguido por dois números hexadecimais (0–9 ou A–F) que representam o valor numérico da codificação do caractere. Por exemplo, o caractere ASCII a til (ã) (cujo valor decimal é 227) é representado por "=E3". Todos os caracteres, com exceção dos caracteres imprimíveis ASCII e o fim-de-linha, devem ser codificados dessa forma. Os caracteres imprimíveis ASCII com valores decimais entre 33 e 126, com exceção do 61 ("="), podem ser representados por si mesmos. Os caracteres tab e espaço também podem ser representados por si mesmos, a não ser que estejam no final de uma linha, caso em que devem ser representados por "=09" e "=20" respectivamente.

Na Figura 4 segue um exemplo de um trecho de uma mensagem de correio eletrônico com uma imagem JPEG[16] anexada:

```
From: "Remetente" <remetente@servidor.com.br>
To: "Destinatario" <destinatario@servidor.gov.br>
Subject: Fotografia
Date: Sun, 30 Jul 2006 22:50:50 -0300
MIME-Version: 1.0
Content-Type: multipart/mixed;
        boundary="----=_NextPart_000_0003_01C6B42A.9A8DD3D0"
X-Priority: 3
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express 6.00.2900.2180
X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2900.2180

This is a multi-part message in MIME format.

------=_NextPart_000_0003_01C6B42A.9A8DD3D0
Content-Type: text/plain;
        format=flowed;
        charset="iso-8859-1";
        reply-type=original
Content-Transfer-Encoding: 7bit

Segue foto anexa.

------=_NextPart_000_0003_01C6B42A.9A8DD3D0
Content-Type: image/jpeg;
        name="Foto.jpg"
Content-Transfer-Encoding: base64
Content-Disposition: attachment;
        filename="Foto.jpg"

/9j/4AAQSkZJRgABAgEBLAEsAAD/4RdvRXhpZgAASUkqAAgAAAAMAA4BAgALAA
AAngAAAA8BAgAG
AAAAqQAAABABAgALAAAArwAAABIBAwABAAAAAQAAABoBBQABAAAAugA
AABsBBQABAAAAwgAAACgB
AwABAAAAAgAAADEBAgAUAAAAygAAADIBAgAUAAAA3gAAABMCAwABAAA
AAgAAAJiCAgAPAAAA8gAA
AGmHBAABAAAABAEAAPQCAAAgICAgICAgICAgAE5JS09OAENPT0xQSVggTDQ
ALAEAAAEAAAAsAQAA
AQAAAEFkb2JlIFBob3Rvc2hvcCA3LjAAMjAwNjowNzozMCAyMjo0Nzo1NQBDb3B5c
mlnaHQgMjAw.....
```

Figura 4: trecho de mensagem convertida pelo MIME

16 Joint Photographic Experts Group – método de compactação de imagens.

A fotografia foi codificada usando a BASE 64. A imagem codificada começa em "/9j/4AA..." Somente uma parte da fotografia codificada foi exibida acima, por se tratar de um arquivo muito grande.

2.5 PROBLEMAS COM O CORREIO ELETRÔNICO

Alguns problemas do correio eletrônico não são novos - eles acompanham a aplicação desde a sua criação mas, naquela época, não eram tão evidentes pois as redes eram menores e seu uso mais restrito (era um uso mais acadêmico – a *Internet* tem sua origem em um projeto do Departamento de Defesa do governo Norte Americano, que visava integrar unidades de pesquisa e universidades - sem o viés comercial que existe hoje). Outros surgiram com a massificação do uso, que fez com que pessoas mal-intencionadas vissem no correio eletrônico uma ferramenta para atingir seus fins (fraude, invasão de privacidade, etc). Até mesmo alguns problemas do correio tradicional migraram para o correio eletrônico, como é o caso dos pedidos de contribuição em épocas como o natal, por exemplo, e o envio de boletos bancários por serviços não prestados. No correio tradicional isso tem um custo (postagem), mas no correio eletrônico basta que se tenha uma lista de destinatários, listas estas que estão se tornando moeda de troca no mercado negro dos *hackers*[17]. Para uma melhor compreensão dos problemas que hoje afetam o correio eletrônico, esses foram divididos em dois grupos:

- Problemas relacionados à arquitetura, que estão ligados à forma como o correio eletrônico foi projetado e implementado.
- Problemas de uso (uso mal-intencionado), que estão relacionados à forma como o correio eletrônico é utilizado.

2.5.1 Problemas na Arquitetura

Para entender os problemas na arquitetura, é preciso lembrar que o correio eletrônico de hoje descende de redes do tipo *store-and-foward*, como a FIDONET [8, 9,10] e a BITNET

[17] Programadores maliciosos que agem com o intuito de violar ilegal ou imoralmente sistemas informatizados [62].

[12]. Os computadores ligados a essas redes não ficavam conectados todo o tempo; portanto, quando uma máquina intermediária era ligada essa se conectava a outra e todas as mensagens cujo caminho passava por àquela eram transferidas, levando-as para mais perto de seus destinatários finais (modelo *store-and-foward*).

Algumas destas redes de troca de mensagens eram acadêmicas (como a BITNET, por exemplo) com relativamente poucas máquinas (se comparadas aos quinhentos milhões de hoje [70]) e relativamente poucos usuários (se comparados aos mais de um bilhão de hoje [71]). Portanto, não havia necessidade de um sistema de segurança elaborado. Uma explicação mais detalhada da história e arquitetura do correio eletrônico se encontra na seção 2.1.

- Segurança

O correio eletrônico da *Internet* funciona sobre o protocolo SMTP, que não é um protocolo seguro. Quando ele foi idealizado não havia necessidade de uma segurança mais elaborada; não havia necessidade de validar os usuários; não havia necessidade de criptografar as mensagens. O modelo do SMTP é baseado na cooperação e confiança entre servidores (ver seção 2.1). Esse modelo supõe que os servidores são bem comportados e não inundariam a rede com mensagens falsas ou mal-intencionadas.

Tudo isso mudou nos anos 90, com o aumento explosivo do número de usuários da *Internet* e a diversificação de seus usos (deixou de ser uma rede puramente acadêmica e se tornou também, uma rede que permite o uso de aplicações comerciais e de lazer). Hoje, essas falhas de segurança do SMTP são muito evidentes e colocam em risco todo o tráfego de mensagens de correio eletrônico.

Algumas das principais características do SMTP que podem ser exploradas como falhas de segurança são [3]:

- As mensagens são enviadas em texto legível, sem criptografia.
- Os endereços do remetente e do destinatário podem ser facilmente descobertos: basta interceptar a mensagem e ler o cabeçalho.
- Não há como saber quanto tempo uma mensagem irá levar para chegar ao seu destinatário, pois não é possível saber quão ocupados os servidores estão, quanto tráfego há na rede, quantas máquinas estão desligadas para manutenção, etc.

- As mensagens podem ficar armazenadas em servidores por um grande período de tempo e vários desses servidores pertencer a terceiros, não havendo como controlá-los.

- Os protocolos POP e IMAP, usados pelo destinatário para ver suas mensagens, também usam texto legível, sem criptografia.

- As mensagens são armazenadas nos servidores em texto legível. Qualquer um com acesso a essas máquinas pode lê-las.

Essas características do modelo SMTP/POP - IMAP podem ser exploradas das seguintes formas [3]:

- **Escuta**: é relativamente fácil para qualquer um com acesso às redes e máquinas por onde as mensagens trafegam capturá-las e lê-las.

- **Roubo de identidade**: como as informações de autenticação (*login* e senha) são enviadas como texto desprotegido, é possível capturar estas informações, ter acesso ao servidor de correio e ler as mensagens nele armazenadas.

- **Invasão de privacidade**: pela informação passada junto com a mensagem é possível saber (pelo endereço IP) a cidade de onde partiu a mensagem e, em alguns casos, até o endereço físico da máquina de onde foi enviada a mensagem [72, 73].

- **Modificação de mensagens**: como as mensagens são armazenadas nos servidores SMTP antes de serem enviadas, qualquer um com acesso a eles pode não só ler as mensagens, mas também modificá-las ou apagá-las. Destinatário e remetente não têm como saber se isso foi feito.

- **Mensagens falsas**: é fácil fabricar uma mensagem e enviá-la como se fosse outra pessoa.

- ***Backups* desprotegidos**: as mensagens armazenadas nos servidores podem ser copiadas e guardadas indefinidamente. Essas mensagens continuarão existindo e poderão ser utilizadas indevidamente muito depois do destinatário ter apagado a cópia que recebeu.

- **Negativas de autoria**: como é fácil criar uma mensagem e enviar como se fosse outra pessoa, é possível enviar uma mensagem verdadeira e depois negar a autoria (ou a resposta). Remetente e destinatário não terão como saber onde está a verdade.

2.5.2 Problemas de Uso

As falhas de segurança citadas no item anterior oferecem inúmeras oportunidades a pessoas mal-intencionadas. As mais comuns são o envio de *spam*, vírus, *worms*, cavalos de tróia e *phishing* (descritos abaixo) que se propagam pela *Internet* trazendo prejuízos aos usuários.

- *Spam*

O termo *spam* refere-se ao envio de mensagens comerciais, pirâmides e afins, sem prévia solicitação e sem que os usuários desejem recebê-las. Normalmente partem de poucos endereços (ou um só) e são copiados para milhares de destinatários.

Apesar de parecer óbvia, a distinção do que é ou não *spam* nem sempre é clara. Segundo a *Mail Abuse Prevention System* (MAPS), uma mensagem eletrônica é *spam* se [4]:

- A identidade e características pessoais do destinatário são irrelevantes pois o conteúdo da mensagem se aplica igualmente a várias pessoas.
- O destinatário não deu, deliberadamente, permissão explícita (que ainda pode ser revogada) para que fosse enviada.
- A transmissão da mensagem parece, para o destinatário, oferecer um benefício desproporcionalmente maior para o remetente.

Pode-se concluir, pela definição acima, que o que é ou não *spam*, depende de uma interpretação pessoal de cada um (cada um deve decidir se o conteúdo da mensagem é ou não relevante e se a vantagem para o emissor é desproporcionalmente maior). Mas uma coisa é certa: o termo *spam* só faz sentido quando aplicado a mensagens enviadas para vários

destinatários (*bulk mail*), pois, quando enviamos um correio eletrônico para alguém pela primeira vez, tecnicamente essa mensagem não foi explicitamente autorizada. O *spam* é um problema por várias razões:

- **É difícil acabar com ele, pois muitas vezes não se consegue localizar a origem:** *spammers* (pessoas ou empresas que enviam *spam*) utilizam endereços eletrônicos fictícios ou criados especificamente para mandar uma carga de mensagens e depois desativam a conta. Como a mensagem já está em curso, a desativação da conta não afeta a sua entrega.

- **Consome recursos compartilhados**: o tráfego de *spam* força os provedores de *Internet* (ISP) a comprar mais largura de banda para suportar o aumento do volume das mensagens.

- **Penaliza duplamente o usuário**: como dito acima, o aumento do tráfego força os provedores a comprar mais largura de banda. Esse custo adicional é repassado aos usuários – que já são vítimas do *spam*. Mas, para o *spammer*, o custo é muito baixo.

- **Consome recursos privados**: se uma empresa tem 100 funcionários e estes passam 10 minutos por dia lendo e apagando *spam*s, no final de um mês o tempo total perdido pela empresa será de, aproximadamente, 360 horas de trabalho.

- **Custo não mensurável**: não se pode medir a irritação e da frustração das pessoas ao abrirem seus correios eletrônicos e descobrirem várias mensagens de *spam*.

- Vírus, *Worms* e Cavalos de Tróia

O correio eletrônico pode ser usado para difundir vírus, *worms* e cavalos de tróia, que são programas que executam a partir da máquina de um usuário e que podem trazer todo tipo de problemas, desde simples inconvenientes (uma janela *pop-up* com um anúncio que fica aparecendo) até graves complicações, que obriguem à formatação do disco rígido da máquina, causando a perda de dados do usuário.

A diferença entre eles é [5]:

- O vírus é um código de computador, escrito com a intenção explícita de se autoduplicar, que se anexa a um programa ou arquivo hospedeiro para poder se espalhar entre os computadores, infectando-os à medida que se desloca. Os vírus podem causar danos aos arquivos do usuário, afetando a confiabilidade e usabilidade do sistema.

- Um *worm*, assim como um vírus, cria cópias de si mesmo de um computador para outro, mas faz isso automaticamente, já que é um programa autônomo e não depende de outros programas para funcionar (ao contrário do vírus). Primeiro ele controla recursos no computador que permitem o transporte de arquivos ou informações e depois contamina o sistema, passando a se deslocar sozinho. O grande perigo dos *worms* é a sua capacidade de se replicar em grande número. Por exemplo, um *worm* pode enviar cópias de si mesmo a todas as pessoas que constem em um catálogo de endereços, e os computadores dessas pessoas passam a fazer o mesmo, causando um efeito dominó de alto tráfego, que pode tornar mais lentas as redes corporativas e a *Internet* como um todo. Quando novos *worms* são lançados, eles se alastram muito rapidamente, obstruindo redes e, muitas vezes, fazendo com que os usuários tenham de esperar mais tempo para acessar sítios na *Internet*. Como os *worms* não precisam viajar através de um programa ou arquivo hospedeiro, eles também podem se infiltrar no sistema do usuário e permitir que outra pessoa controle o seu computador remotamente.

- Assim como o mitológico cavalo de Tróia parecia ser um presente, mas na verdade escondia soldados gregos em seu interior que tomaram a cidade de Tróia, os cavalos de Tróia da atualidade são programas de computador que parecem ser úteis, mas comprometem a segurança e causam muitos danos. Um cavalo de Tróia recente apresentava-se como um correio eletrônico com anexos de supostas atualizações de segurança da Microsoft, mas era um programa, que tentava desativar programas antivírus e *firewalls*[18].

[18] *Firewall*: dispositivo de uma rede de computadores que regula o tráfego de dados entre redes distintas e impede a transmissão e/ou recepção de dados nocivos ou não autorizados de uma rede a outra.

- *Phishing*

O *phishing* é uma forma de engenharia social (técnica usada para se obter informações sigilosas através da enganação ou da exploração da confiança das pessoas). No *phishing* o usuário recebe uma mensagem aparentando ser de uma entidade ou empresa legítimas (a Receita Federal ou um banco, por exemplo) e, nessa mensagem, é solicitado que ele envie dados sigilosos (senhas, números de contas, etc.).

2.6. SOLUÇÕES DISPONÍVEIS PARA OS PROBLEMAS DO CORREIO ELETRÔNICO

Os problemas descritos no capítulo anterior chegaram a um ponto em que ameaçam a própria viabilidade do correio eletrônico. Para resolver ou atenuar alguns desses problemas foram surgindo soluções (a maioria delas voltada para um problema específico). Para que o usuário se proteja, ele terá que ter um conjunto de soluções instaladas em sua máquina ou esperar que seu provedor de acesso as tenha. Abaixo apresentamos algumas dessas soluções (as mais comuns), divididas de acordo com o problema que se propõem resolver.

2.6.1 Soluções contra o *spam*

Atualmente o combate ao *spam* é feito, principalmente, de duas formas: filtros de *spam* e DNS *Blacklists*.

- Filtros de *spam*

Filtros de *spam* são programas que eliminam as mensagens classificadas como *spam*. Esses filtros podem estar localizados na máquina do usuário ou em seu servidor de correio eletrônico. O problema com esse tipo de *software* é que, muitas vezes, mensagens legítimas acabam classificadas como *spam*.

Os tipos de filtros de *spam* mais utilizados são:

- ***Integrated, Internet-based spam filters***: são serviços oferecidos por terceiros e, geralmente, pagos. O banco de dados com as definições do que é *spam* é remoto e comum para todos os usuários. Essas definições são baseadas em palavras ou frases comumente encontradas em mensagens de *spam* ("grátis" e "oferta", por exemplo) e também podem ser atualizadas pelos próprios usuários, que classificam as mensagens que recebem em *spam* ou não e informam ao provedor do serviço. Isso gera um problema, pois o que é *spam* para uns pode não ser para outros (alguns usuários, por exemplo, quando não querem mais receber mensagens de uma determinada lista, classificam a lista como *spam*, prejudicando os demais membros do grupo que também deixarão de receber as mensagens). O número de mensagens legítimas erroneamente classificadas com *spam* tende a ser alto.

- ***Integrated, algorithmic spam filters***: utilizam algoritmos para determinar o que é *spam*. Um dos mais usados é o algoritmo Bayesiano[19], que procura estatisticamente por palavras que aparecem em *spam*s e utiliza essas estatísticas para classificar as mensagens que são recebidas. Para se adaptar a isso os *spammers* têm trocado texto por imagens.

- ***Proxy spam filters***: um *proxy*[20] é um programa situado entre a máquina do usuário e seu servidor de correio eletrônico, e que pode estar localizado na própria máquina do usuário ou em seu servidor de correio eletrônico. Uma vez instalado e configurado, o *proxy* se torna um intermediário entre o servidor de correio e o UA do usuário, filtrando as mensagens. As definições do que deve ser classificado como *spam* são feitas pelo próprio usuário (caso o *proxy* esteja em sua máquina, ou pelos responsáveis pelo servidor de correio, caso o *proxy* esteja instalado no servidor).

[19] Thomas Bayes – matemático inglês do século 18, que introduziu uma nova visão sobre probabilidades em um ensaio publicado após a sua morte em 1761 [74].

[20] O termo *proxy* em TI pode se referir a um serviço, uma máquina ou um programa, geralmente designando algo que atue como intermediário.

- ***Server-side spam filters***: são filtros localizados no servidor de correio eletrônico. Dependendo da configuração, podem apagar diretamente as mensagens classificadas como *spam* ou apenas marcá-las como *spam* e deixar que o usuário lide com elas como preferir.

Algumas empresas (como a Microsoft, por exemplo), estão trabalhando em filtros inteligentes, que seriam capazes de aprender e acompanhar as mudanças de táticas dos *spammers*. Outros filtros utilizam um sistema de confirmação [54] (uma mensagem é enviada ao remetente para que este confirme o envio da mensagem original) que efetivamente impede que mensagens enviadas por robôs cheguem ao destinatário.

Filtros anti-*spam*, contudo, não são uma unanimidade. Muitas pessoas os consideram uma invasão de privacidade. O sociólogo Silvio Lefevre, em um artigo publicado[17] diz:

> "Estão violando e censurando nossa correspondência (...) a maioria dos provedores, a pretexto de combater o que chamam de "*spam*", censura todos os *e-mails*. Nerds travestidos em censores inventaram programinhas para bloquear e-mails, que assim nem chegam a você.".

- DNS *Blacklists*

DNS *Blacklists* são listas de nomes de domínios de DNS[21] de onde partem *spams*, usadas para barrar mensagens vindas desses domínios, produzidas e colocadas na *Internet* por organizações (DBSL.org [18], rfc-ignorant.org [19], entre outras) e utilizadas por vários provedores de acesso à *Internet*. O grande problema com essas listas é que domínios legítimos podem ser incluídos por engano – existem vírus como o MyDoom, por exemplo, que copiam os endereços IP das máquinas infectadas e enviam mensagens em nome delas, fazendo com que o domínio tenha seu nome incluído na lista injustamente - ou de forma mal-intencionada, causando enormes prejuízos. A validade das listas está diretamente ligada à seriedade da organização, que deve atualizá-las com freqüência.

[21] Domain Name System – sistema utilizado na Internet para traduzir nomes de servidores em endereços de IP.

Algumas empresas estão desenvolvendo filtros adaptativos [20] e trabalhando em tecnologias que permitirão identificar o remetente das mensagens, através da autenticação do remetente. Essa autenticação é feita da seguinte forma [21]:

1) Informação de autenticação para o servidor que envia correio eletrônico, colocada no DNS (ex: uma chave pública para assinatura)

2) O servidor que recebe a mensagem compara as referências do servidor emitente com a informação do servidor contida no DNS, e assim valida o servidor.

Outros mecanismos para desencorajar o *spam* são a cobrança por mensagem enviada e a criminalização do *spam*.

Algumas pessoas (como Suzanne Sluizer – co-autora do MTP – *Mail Transfer Protocol*, o antecessor direto do SMTP) defendem que seja escrito um novo protocolo para eliminar as falhas de segurança do SMTP [22]. Um dos projetos é o IM2000 (ver seção 2.7.1). Esse caminho é diferente da solução apresentada nesta dissertação, pois, como o SMTP é um padrão de enorme aceitação, acreditamos ser melhor mantê-lo o mais inalterado possível, e tentar corrigir os problemas advindos da sua falta de segurança através da limitação do que é transportado por ele.

Soluções menos drásticas para outros problemas do SMTP passam por estender suas funcionalidades. A RFC 3207, por exemplo, propõe uma extensão que permite que clientes e servidores SMTP usem TLS – *Transport Layer Security* para prover uma comunicação segura e autenticada na *Internet* [24].

As soluções descritas acima, apesar de válidas, atuam em pontos específicos, e muitas vezes, dependem da colaboração de governos (como criação de legislação pertinente), provedores (instalação de filtros) e até dos usuários (manter os seus antivírus atualizados). A solução proposta difere delas por atacar os vários problemas de uma forma simples e integrada que, uma vez implementada, funcionará por si só, não requerendo manutenção periódica, como filtros e bases de dados (e até legislação!). Além disso, apesar de não envolver mudanças no protocolo propriamente dito (SMTP), tal proposta muda o modelo em que é utilizado, fazendo com que suas falhas de segurança sejam mais difíceis de serem exploradas.

2.6.2 Soluções contra Vírus, *Worms* e Cavalos de Tróia

Atualmente a solução mais utilizada é o uso de programas antivírus, que devem ser instalados e mantidos atualizados pelos próprios usuários. O uso de *firewalls* também é recomendado, pois eles podem impedir que programas não explicitamente autorizados pelo usuário, como cavalos de tróia, por exemplo, consigam enviar dados sigilosos do computador do usuário para fora (*Internet*).

Paralelamente, a disciplina do próprio usuário é muito importante, não abrindo mensagens de remetentes que ele não consiga identificar. Essa disciplina, porém, tem alcance limitado, pois como vimos, é possível mandar mensagens de correio eletrônico como se fosse outra pessoa.

2.6.3 Soluções contra o *phishing*

No momento, a melhor defesa contra o *phishing* é a disciplina do próprio usuário, não executando programas anexados e verificando com a instituição ou empresa se aquela mensagem é legítima.

2.7 ALTERNATIVAS AO CORREIO ELETRÔNICO ATUAL

Os problemas do atual modelo de correio eletrônico (ver seção 2.5) vêm sendo estudados por pesquisadores ao redor do mundo e algumas propostas têm surgido para minorar seus efeitos. Estas propostas podem ser divididas em três grupos, de acordo com o que propõe: 1- substituição do SMTP, 2- extensões ao SMTP e 3- preservação do SMTP, modificando a forma de transportar das mensagens ou utilizando outros protocolos em conjunto com o SMTP.

2.7.1 Substituição do SMTP

Bernstein [26] propõe a criação de uma nova infra-estrutura, chamada *Internet Mail* 2000 (IM2000), cujo ponto principal seria tornar o armazenamento das mensagens enviadas responsabilidade do remetente e não mais do destinatário. Isso seria obtido através do uso de quatro novos protocolos:

- ***Message Store Originator Access Protocol*** - responsável pela comunicação entre o UA do remetente e o servidor de armazenamento das mensagens.

- ***Message Store Recipient Access Protocol*** - responsável pela comunicação entre o UA do destinatário e o servidor de armazenamento.

- ***Recipient Notification Agent Submission Protocol*** – envia notificações de mensagem ao agente de notificação de mensagens.

- ***Recipient Notification Agent Query Protocol*** – protocolo de comunicação entre o UA do destinatário e o agente de notificação de mensagens.

O IM2000 (ver Figura 5) introduz ainda, um servidor de armazenagem, que guarda as mensagens enviadas, e um notificador de mensagens recebidas, que informa ao destinatário que existem mensagens para ele no servidor de armazenagem.

Essa proposta é similar, no conceito, à proposta apresentada nesta dissertação: transferir o ônus da armazenagem para o remetente e criar uma mensagem-resumo, que notificaria o destinatário sobre mensagens recebidas, para que ele pudesse decidir se quer ver a mensagem completa ou não. A diferença está na implementação: enquanto no IM2000 existe a criação de novos protocolos e uma ruptura com o modelo atual, a nossa proposta aproveita a arquitetura utilizada hoje e obtém o mesmo resultado através da criação de dois processos, que seriam executados nos servidores de correio eletrônico sem interferir com o funcionamento regular do SMTP.

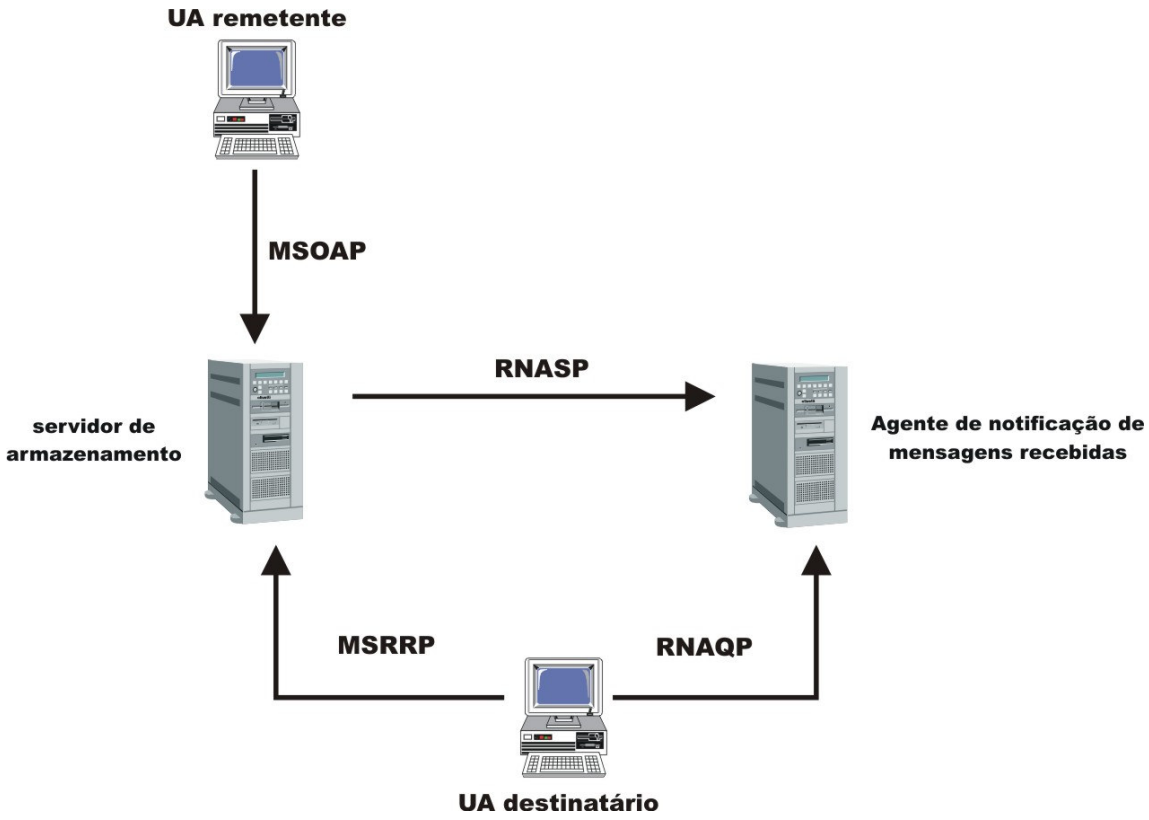

Figura 5: IM 2000

Weinman [27] propõe a criação do AMTP - *Authenticated Mail Transfer Protocol* como uma alternativa ao SMTP. O AMTP utiliza TLS (*Transport Layer Security*) [58] para criar um ambiente confiável entre servidores envolvidos em uma troca de mensagens. Esses servidores, então, trocam informações de segurança (MPC - *Mail Policy Codes*) para estabelecer permissões de envio/recebimento de mensagens.

O AMTP cria uma relação de confiança entre MTA's. Essa relação requer um servidor identificado de cada lado da conexão, para ser contatado em caso de violação das políticas de segurança. É importante frisar que o usuário não necessita estar autenticado. Basta que seu provedor esteja. A autenticação é feita através de certificados X.509 [28] emitidos por uma entidade certificadora.

Outro ponto importante do AMTP são as MPC's - *Mail Policy Codes*, que estabelecem políticas de classificação para as mensagens, permitindo que MTA's aceitem ou recusem mensagens com base nessas políticas.

Essa proposta difere radicalmente da apresentada nesta dissertação, pois não só propõe uma ruptura completa com o modelo atual, mas também introduz a necessidade de certificados de autenticação, o que pode aumentar consideravelmente os custos para os usuários.

2.7.2 Extensão do SMTP

Zhenhai, Kartik e Yingfei [25, 61], da Florida State University, propõem a criação de um novo protocolo, chamado *Differentiated Mail Transfer Protocol* (DMTP), que estenderia o SMTP, permitindo a classificação dos servidores de envio de e-mail (MTA's) em três níveis: *Allowed* (ou *trusted*), *Denied* (ou *black-listed*) e *Unclassified* (ou *untrusted*). As mensagens oriundas dos servidores classificados como *trusted* seriam encaminhadas normalmente, como é feito hoje. As mensagens vindas de servidores classificados como *denied* seriam recusadas e não chegariam ao destinatário. As mensagens provenientes de servidores *unclassified* seriam retidas no próprio servidor de origem, que encaminharia ao destinatário um aviso (*intention message*) indicando que existe uma mensagem para ele. Com base nesse aviso, o destinatário pode decidir se deseja ou não receber a mensagem.

Para esse fim, os autores do DMTP propõem a criação de mais dois comandos para o SMTP – MSID e GTML, usados respectivamente para informar um identificador de mensagem e para recuperar esta mensagem. Temos, portanto, a idéia de uma mensagem de notificação que precede a mensagem propriamente dita, que é a base para a implementação do modelo *pull*.

Essa proposta tem algumas similaridades com a apresentada nesta dissertação – a possibilidade de implementação gradual, visto que o modelo proposto é compatível com o atual, e a necessidade de uma presença mais prolongada do remetente (no caso de servidores *unclassified*), já que as mensagens seriam armazenadas no servidor de origem. Em contraste com a presente dissertação, pode ser apontado o fato de ser uma proposta voltada unicamente ao combate do *spam* e a necessidade de alterações (ainda que pequenas) no SMTP.

Conforme dito em [60], desde sua proposição inicial em 1982, o conjunto de comandos do protocolo SMTP foi ampliado em diversas ocasiões, até ser novamente consolidado em 2001 [2]. Nesta consolidação o método de extensão proposto na RFC 1869

[59] foi incorporado e, ainda hoje, serve como base para a adição de novas funcionalidades ao SMTP.

Extensão	Descrição	Definida em
8BITMIME	Permite transmissão de dados codificados em 8 bits	RFC 1652
AUTH	Permite autenticação de cliente e servidor SMTP	RFC 2554
CHUNKING	Melhora a eficiência no envio de arquivos grandes	RFC 3030
PIPELINING	Permite o envio de uma seqüência de comandos	RFC 2920
SIZE	Implementa a declaração de tamanho da mensagem	RFC 1870
STARTTLS	Implementa *Transport Layer Security*	RFC 3207

Tabela 3: algumas extensões do SMTP [60]

Ao comando de saudação original do SMTP (HELO) foi acrescentado o comando alternativo EHLO. Ao receber o comando EHLO, o servidor deve responder como a lista de extensões que é capaz de suportar. Uma lista de extensões populares é apresentada na Tabela 2 que demonstra que o método de extensões tem sido usado para atacar uma série de problemas do correio eletrônico. Exemplificando, as extensões STARTTLS e AUTH aumentam a segurança do SMTP, ao passo que PIPELINING melhora sua eficiência. Tratando especificamente da eficiência na transmissão de grandes mensagens temos SIZE e CHUNKING, importantes para o envio de conteúdos volumosos, como áudio ou vídeo.

Finalmente, a extensão 8BITMIME dá suporte à codificação em oito bits, vencendo a antiga limitação de codificação em sete bits e aumentando a eficiência no transporte das mensagens. Portanto, podemos afirmar sem grandes riscos, que as extensões são um fator que contribui para a longevidade do SMTP.

A principal desvantagem das soluções baseadas em extensões do SMTP é a necessidade de alteração de servidores e clientes de correio eletrônico, ainda que de forma menos extrema do que as propostas apresentadas na seção anterior.

2.7.3 Preservação do SMTP

T. J.Kim [53], do Cheonan National Technical College, na Coréia do Sul, apresenta uma proposta parecida com a desta dissertação, isto é, as mensagens ficam armazenadas no servidor de saída e uma notificação é enviada ao destinatário para que este decida se quer ver ou não a mensagem original.

A proposta por ele apresentada difere, no entanto, do presente modelo por não prever a possibilidade de criptografar as mensagens, por não apresentar uma solução para o problema das listas (uma mensagem destinada a várias pessoas) e por criar um servidor (*Downemail Destination Server*) que receberia os avisos de recebimento de mensagem, mas que não poderia lidar com as mensagens originais, impedindo um esquema de "listas brancas" como o proposto no seção 4.1.

David Turner e Keith Ross [55], ao tratar do problema de *Continuous Media E-mail* (CM E-mail), propõem que a mídia também fique armazenada no servidor de saída (do remetente).

CM E-mail são mensagens de correio eletrônico que contém uma mídia contínua (que é transferida e exibida simultaneamente), como vídeo ou áudio, ao contrário do texto ou de fotografias, que são mídias estáticas. CM E-mail apresenta algumas características que o difere de mensagens com mídias estáticas:

- O tamanho das mensagens tende a ser maior, devido às características da mídia (arquivos de som e vídeo são maiores e menos comprimíveis que arquivos texto – formatos como o MPEG[22] e o MP3[23], por exemplo, já são compactados e qualquer compactação adicional, surte pouco efeito), portanto, o espaço necessário para armazenagem dessas mensagens deve ser maior.

- A tolerância a retardos de transmissão é menor (pode haver atrasos grandes na renderização da mensagem, principalmente em enlaces com baixas taxas de transmissão).

[22] Moving Picture Experts Group – grupo de trabalho da ISO/IEC encarregado de desenvolver padrões de codificação de áudio e vídeo.

[23] MPEG-1 Audio Layer 3 – formato de codificação de arquivos de áudio que utiliza compressão com perdas.

Pela proposta por eles apresentada, a mensagem ficaria armazenada no servidor de saída e uma notificação (similar à mensagem-resumo proposta nessa dissertação) seria enviada ao destinatário, que, caso desejasse, veria o conteúdo via *streaming*[24].

A proposta aqui referenciada é similar à apresentada nesta dissertação, e propõe a mesma solução, isto é, o armazenamento das mensagens pelo remetente e a recuperação do conteúdo por requisição explícita do destinatário. A diferença está no foco: enquanto a proposta dessa dissertação é geral, para todo tipo de mensagens, a deles foca, exclusivamente, no problema de mensagens com mídia contínua.

2.8 CONCLUSÃO

O correio eletrônico é uma aplicação de redes que permite que pessoas enviem mensagens entre si. Sua origem está nos programas de BBS como a FIDONET, em programas de envio de arquivos como o UUCP e em redes acadêmicas como a BITNET. O principal protocolo utilizado é o SMTP, responsável pelo envio das mensagens através da rede. Apesar de ser um protocolo robusto e confiável, o SMTP mostra sinais da passagem do tempo e tem sido complementado para se manter viável. Para ler as mensagens o destinatário utiliza os protocolos POP ou IMAP. O POP é um protocolo mais simples, que permite que usuários com conexões esporádicas façam o *download* de mensagens para suas máquinas; já o IMAP é um protocolo mais sofisticado, e permite que os usuários manipulem suas mensagens diretamente no servidor. Inicialmente o correio eletrônico oferecia suporte apenas a mensagens de texto, mas depois, com o MIME, passou a ser possível enviar quase todo o tipo de arquivos eletrônicos anexados a uma mensagem.

O correio eletrônico (e seus protocolos, principalmente o SMTP) permaneceu relativamente inalterado durante o tempo. Apesar das várias extensões propostas ao longo dos anos, seu funcionamento básico continuou o mesmo. Isso propiciou o aparecimento de problemas para os quais sua arquitetura não oferece respostas satisfatórias e problemas de uso não antevistos por seus idealizadores.

Os problemas na arquitetura decorrem, principalmente, de sua origem nas redes acadêmicas, compostas de (relativamente) poucas máquinas e onde havia um controle maior

[24] Tecnologia de transmissão de mídia contínua, que é vista (ou ouvida) enquanto é transmitida.

sobre quem era usuário. Portanto, não havia a necessidade de uma segurança mais elaborada: a segurança era baseada na confiança que existia entre os participantes.

Os problemas de uso começaram a surgir quando a *Internet* se popularizou, perdendo sua característica acadêmica, e se transformou em uma rede também comercial. Nesse ponto surgiu o *spam*, que pode ser apontado como um dos principais problemas do correio eletrônico hoje.

Com a massificação do uso, alguns problemas (decorridos de sua arquitetura inicial e do uso distorcido da aplicação) ficaram evidentes – *spam*, vírus e falta de segurança, entre outros, e algumas soluções foram propostas para resolver ou minimizar esses problemas. Estas soluções podem ser independentes do SMTP (filtros de *spam*, programas anti-virus, etc) - a maior parte destas soluções, porém, requer uma participação ativa e vigilante dos usuários, o que nem sempre é possível, pois existe uma grande variedade de usuários com níveis de conhecimento técnico heterogêneos. Os usuários mais avançados, que têm mais conhecimentos técnicos, podem se proteger melhor; já os leigos e inexperientes geralmente ficam dependendo que outras pessoas configurem suas máquinas (e muitas vezes têm que pagar por isso) ou ficam entregues à própria sorte -, extensões ao SMTP (como a criação do protocolo DMTP, que estenderia o SMTP) ou a criação de novos protocolos para substituir o SMTP, como o IM 2000, por exemplo. Quaisquer que sejam as soluções propostas é evidente que o SMTP ficou ultrapassado e, sozinho, não consegue mais oferecer a segurança necessária para a troca de mensagens de correio eletrônico.

3. A ARQUITETURA PROPOSTA

Neste capítulo descreveremos a arquitetura proposta para enfrentar os problemas citados anteriormente, que é composta de dois processos: um para criar e enviar a mensagem-resumo e armazenar a mensagem original; o segundo tem por função entregar a mensagem original (caso ela seja requisitada) e fazer a validação de segurança. Esses dois processos são leves e podem ser implementados de várias formas em diversas linguagens de programação (o protótipo foi feito em Java[25]).

Na arquitetura proposta (ver Figura 6), quando a mensagem chega ao servidor SMTP do remetente e antes de ser colocada em sua caixa de saída (local onde é mantida até que seja enviada), é feita uma cópia do cabeçalho, adicionando-se mais algumas informações que permitirão que o destinatário identifique o remetente, o assunto e se a mensagem tem arquivos anexados permitindo que ele decida se quer ou não ver a mensagem original, e somente essa cópia (mensagem-resumo) é enviada. A mensagem original fica armazenada (em sua forma criptografada) no repositório, que pode ser na mesma máquina onde é feita a divisão ou em uma máquina distinta. O único requisito é que esta máquina seja capaz de atender a requisições HTTP. A mensagem-resumo prossegue via SMTP, como na arquitetura atual (é enviada até o servidor SMTP do destinatário, onde agurdará em sua caixa de entrada até que seja lida ou transferida para o UA do destinatário via POP ou IMAP), até chegar ao

[25] Linguagem de programação orientada a objetos desenvolvida pela Sun Microsystems

destinatário, para que este decida se deseja transferir a mensagem completa ou não. Caso afirmativo, ele envia uma requisição HTTP ao repositório para que encaminhe a mensagem. Caso contrário, ele pode deixar a mensagem lá para que ela seja excluída automaticamente, depois de expirado o prazo de validade ou deixá-la guardada para lê-la em outro momento. Para viabilizar o modelo, dois novos processos (*daemons*) foram desenvolvidos: um para dividir a mensagem (separar o cabeçalho do corpo) e criar e enviar a mensagem-resumo para o destinatário e outro para gerenciar a entrega do corpo. Este último transfere a mensagem original quando esta é requisitada, funciona como "coletor de lixo" para apagar mensagens expiradas e efetua a autenticação de segurança.

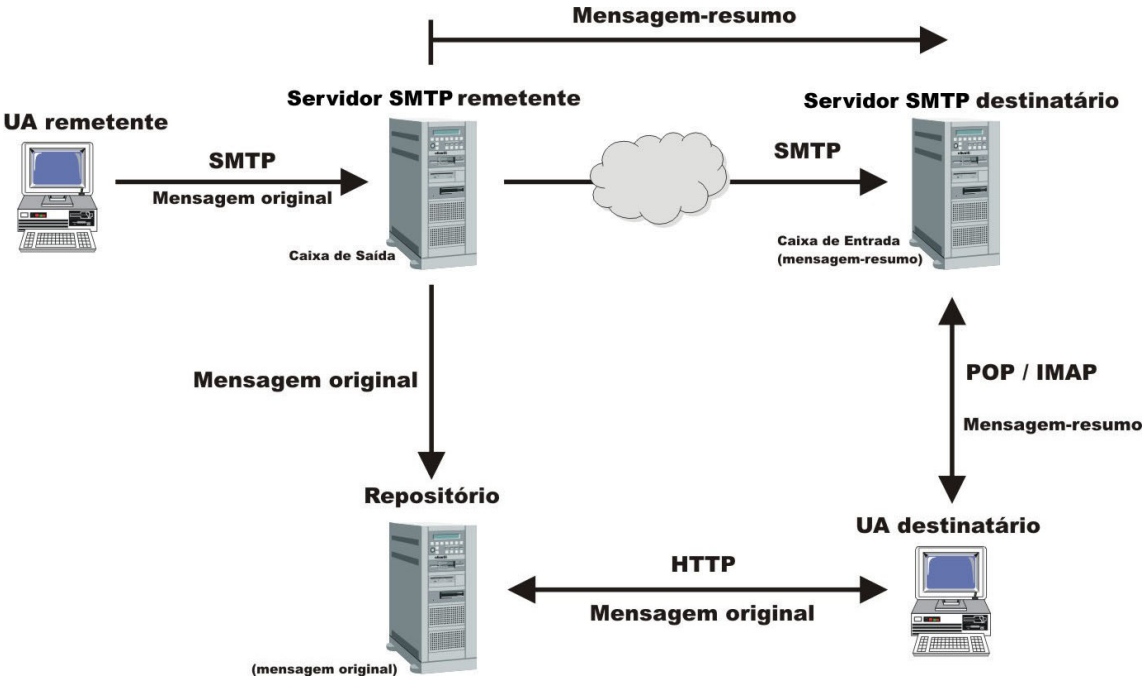

Figura 6: Sistema de Transporte proposto

3.1 DOIS NOVOS PROCESSOS

Para fazer a divisão da mensagem, um novo processo foi criado no MTA do remetente. Esse processo faz uma análise da mensagem original, separando o cabeçalho do corpo. Ao cabeçalho ele adiciona o identificador dessa mensagem, que é usado pelo destinatário para recuperar a mensagem original, que é armazenada no repositório sob a forma

de um arquivo com a extensão ".eml". O mesmo identificador é adicionado à mensagem original e é composto de duas partes: uma é o identificador de armazenamento, para possibilitar a recuperação da mensagem original, e outra usada para criptografar a mensagem original, conforme será explicado adiante, para aumentar a privacidade das mensagens armazenadas. Esse cabeçalho é adicionado a uma nova mensagem, que contém uma lista dos arquivos anexados à mensagem original (se houver) e um *link* HTTP para a mensagem original.

Outro processo foi criado para entregar a mensagem original quando esta for solicitada. Esse processo é executado no repositório, que pode ser uma máquina distinta do servidor SMTP do remetente ou não. Ele faz a autenticação do destinatário através do identificador da mensagem, de modo que este seja o único a ter permissão para ver o seu conteúdo. Para evitar o acúmulo de mensagens, muitas vezes esquecidas pelos destinatários, este processo também executa um "coletor de lixo", que apaga mensagens após um certo tempo. Ao serem enviadas, as mensagens ganham um prazo de validade e após esse prazo elas são apagadas. Como é impossível saber o real número de destinatários (por exemplo um endereço pode ser allUsers@servidor, que na verdade é um grupo de destinatários), as mensagens serão excluídas do servidor apenas com base no prazo de validade já que, se fosse dada permissão para que usuários individuais apagassem mensagens, eles poderiam apagá-las antes que outras pessoas as tivessem lido.

3.2 ARQUIVOS ANEXADOS

Os arquivos anexados são listados na mensagem-resumo que vai para o destinatário. O corpo da mensagem contém um *link* para o(s) arquivo(s), para que estes possam ser vistos.

Os arquivos são reconvertidos para o formato original no servidor de armazenamento (eles estavam codificados pelo MIME). Quando é feita a requisição HTTP eles são enviados e exibidos no navegador ou no programa apropriado (ou então o usuário pode optar por fazer o *download* do arquivo), conforme já acontece hoje quando é feita a solicitação de um arquivo a um servidor HTTP.

3.3 SEGURANÇA

Para aumentar a inviolabilidade da mensagem original enquanto estiver armazenada no repositório, um esquema que autentica o usuário que tenta vê-la e que armazena a mensagem em sua forma criptografada foi desenvolvido. Ao ser criada a mensagem-resumo, um identificador é criado e gravado tanto na mensagem original quanto no *link* que é colocado na mensagem-resumo. A mensagem original é criptografada (para a prova de conceito – ver capítulo 4 - foi utilizado o algoritmo de criptografia DES[26]) antes que seja gravada no disco rígido, de modo que não haja uma versão dessa mensagem sem criptografia no repositório de armazenamento.

A criptografia, como disciplina, é o estudo dos princípios e técnicas pelos quais uma informação pode ser transformada de seu formato original para algo que seja ilegível por quem não tem a chave para reverter o processo, sendo um ramo especializado da teoria da informação. Como técnica, é uma codificação que consiste em usar fórmulas matemáticas para transformar informação em dados (aparentemente) sem sentido, impedindo sua leitura por pessoas que não possuam a chave para descriptografar os dados. Algumas linguagens de programação hoje em dia, já vêm com módulos de criptografia (o JAVA, por exemplo, tem o pacote Javax, utilizado para construir o protótipo usado na prova do conceito – ver capítulo 4), difundindo e facilitando sua utilização. Um dos esquemas de criptografia mais comuns hoje é o de chave pública/privada. Neste esquema duas chaves são utilizadas: uma é pública (pode ser conhecida por outras pessoas), utilizada para criptografar a informação. A outra é privada (só é de conhecimento do destinatário da mensagem). Somente com a chave privada é possível descriptografar os dados, mesmo que se conheça a chave pública. Mais informações sobre criptografia podem ser encontradas em [50], [51] e [52].

Ao clicar no *link* para ver a mensagem original, o identificador é retornado e somente com ele a mensagem pode ser lida. Para aumentar a segurança da mensagem, é possível usar um esquema de chave privada/chave pública. Se o remetente conhecer a chave pública do destinatário, a mensagem é primeiro criptografada usando essa chave e depois a parte do identificador da mensagem é usada para a criptografia. O identificador é um número de 512 bits escolhido aleatoriamente, de forma a dificultar ataques à base de dados (onde um atacante tentaria ler mensagens pedindo mensagens em seqüência).

---

[26] Data Encryption Standard – um dos algorítimos de criptografia mais utilizados no mundo [75].

Com a implementação do *user agent* (ver seção 4.1) esse esquema de criptografia poderá ser opcional (ao redigir a mensagem o remetente indica se aquela mensagem deve ou não ser criptografada, e um novo campo no cabeçalho indicará ao servidor se a criptografia deve ou não ser feita). Isso pode diminuir o *overhead* no servidor (que não terá que criptografar todas as mensagens, mesmo as com conteúdo livre) e também resolverá o problema das listas (ver seção 3.6), onde nem todos os destinatários poderão ter a chave para descriptografar a mensagem.

3.4 ENVIO DA MENSAGEM-RESUMO

Após ser feita a divisão da mensagem, a mensagem-resumo - composta do cabeçalho (com os campos adicionais), um texto indicando o remetente da mensagem e o assunto e o *link* para ver a mensagem original - segue o caminho convencional de um correio eletrônico hoje, isto é, segue via SMTP pela rede até o MTA (servidor SMTP) do destinatário e deste para o servidor POP ou IMAP, de onde será transferida pelo usuário para sua máquina. Como a mensagem-resumo normalmente será bem menor que a mensagem completa (nos testes realizados seu tamanho aproximado foi de 2Kb), isso permite seu envio a terminais com memória limitada, como telefones celulares e PDAs[27], e faz com que seu *download* seja muito rápido, mesmo em enlaces com baixas taxas de transmissão.

3.5 RECUPERAÇÃO DA MENSAGEM ORIGINAL

Para ver a mensagem original, o usuário deve acionar o *link* que acompanha a mensagem-resumo, enviando uma requisição HTTP para a máquina onde está armazenada (máquina que contém o repositório). O identificador é usado para localizar e descriptografar a mensagem que é, então, exibida pelo navegador de *Internet* (*web browser*) que estiver instalado na máquina do destinatário (um *user agent* específico para funcionar com esse novo modelo poderá ser desenvolvido – ver seção 4.1). Se houver arquivos anexados, estes serão exibidos na forma descrita na seção 3.2. Assim, não é necessária qualquer alteração nos clientes usados atualmente, já que estes, em grande parte, já possuem suporte a *link*s HTTP.

[27] PDA: Personal Digital Assistant: computador de mão, também chamado de *palmtop*.

3.6 LISTAS

Muitas mensagens são endereçadas a mais de uma pessoa, na forma de listas de discussão. A maior parte dos sistemas de correio permite a criação de listas, facilitando o envio de mensagens para grupos de usuários. Como a expansão das listas pode ser feita no destino final, não se pode criar uma cópia da mensagem para cada destinatário, pois não há como saber quantos são. Portanto uma única cópia será mantida por mensagem, e essa cópia será apagada ao término do prazo de validade.

O esquema original de criptografia prevê que a chave para descriptografar a mensagem seja embutida no *link*. Nesse caso nenhuma mudança é necessária para acomodar as listas, visto que cada membro da lista poderá descriptografar a mensagem.

Caso seja usado um esquema de chave pública/privada, como (possivelmente) nem todos os destinatários das listas podem ter a chave para descriptografar a mensagem, a criptografia dessas mensagens poderá ser opcional com a implementação do *user agent* (ver seção 4.1), ou então uma chave comum ao grupo poderá ser criada.

3.7 VANTAGENS DA NOVA ARQUITETURA

A arquitetura proposta apresenta as seguintes vantagens:

- **Dificulta o *spam***: o novo modelo dificulta a prática de *spam*, pois somente o cabeçalho das mensagens é automaticamente enviado ao destinatário. As mensagens completas somente serão vistas se o usuário quiser. Além disso, o novo modelo transfere o ônus da armazenagem das mensagens para o servidor do remetente - fica mais difícil e mais caro armazenar milhares de mensagens em seu servidor de saída. Mais ainda: o novo modelo exige que o servidor de envio das mensagens fique *online*, facilitando sua identificação.

- **Diminui o tráfego na Rede**: como somente um pequeno cabeçalho é enviado, o volume de tráfego na *Internet* será reduzido.

- **Caixa de entrada distribuída:** a caixa de entrada do usuário fica distribuída entre vários servidores distintos, ao contrário do IMAP em que todas as mensagens ficam em um mesmo servidor. Dessa forma, as mensagens podem ser lidas de qualquer máquina e, caso aconteça algo com um dos servidores, as mensagens nos outros servidores ainda estarão disponíveis. Isso é possível pois o elo que permite a leitura da mensagem é o *link* HTTP, que pode ser copiado e utilizado a partir de um navegador instalado em qualquer máquina.

- **Diminui o volume de mensagens no servidor de destino:** como as mensagens passam a ser armazenadas no servidor do remetente, o servidor do destinatário ficará menos sobrecarregado, reduzindo o risco do usuário ficar impedido de receber mensagens porque sua caixa de entrada está cheia.

- **Diminui o risco de infecção por vírus, *worms* e cavalos de tróia**: ao ler o cabeçalho o usuário pode ou não identificar o remetente, enquanto o arquivo (possivelmente) infectado ainda está no servidor. Ele tem como decidir não buscar a mensagem ou então lê-la, mas não abrir o anexo. Se o usuário receber um correio eletrônico com vírus, o servidor que originou esse correio eletrônico é facilmente identificado, e pode ser contatado para impedir que outras pessoas recebam o vírus. Apesar disso não impedir a existência de servidores fantasmas, a necessidade de criar uma presença mais permanente do que simplesmente para o envio de correio eletrônico aumenta o custo do *spam* para o remetente.

- **Compatibilidade**: como o modelo proposto utiliza os protocolos SMTP e HTTP, ele é totalmente compatível com o modelo atual e, portanto, pode ser implementado em etapas, sem que os serviços de correio eletrônico em funcionamento hoje tenham que ser interrompidos.

3.8 LIMITAÇÕES DA ARQUITETURA APRESENTADA

A nova arquitetura não elimina completamente o *spam*. Servidores podem ser criados para a distribuição de correio eletrônico, da mesma forma que pornografia é colocada em servidores de acesso livre da *Internet*. Como o modelo pode conviver com o antigo, *spammers* também podem usar o SMTP até que sejam dadas aos usuários ferramentas para só aceitar

correio eletrônico se este for do tipo “somente cabeçalho”. O uso de HTTP permite que vírus e *worms* que explorem as falhas de segurança dos leitores de HTML continuem a ser usados.

O modelo gera uma carga maior nos servidores de envio de correio eletrônico, que também teriam a função de armazenar as mensagens até que expirem. O “prazo de validade”, apesar de reduzir o problema anterior, também pode gerar outro problema, porque os usuários podem perder mensagens - se saírem de férias por períodos longos, por exemplo. Tem de ser criado um mecanismo similar ao auto-respondedor “*vacation*”, que permita ao usuário manter as mensagens ativas até seu retorno (ou então receber todas as mensagens ou, pelo menos, as mensagens que estão para expirar enquanto ele estiver de férias).

Finalmente, pode ser que nem todos os usuários fiquem satisfeitos em ter mais um passo entre receber a notificação de correio eletrônico e receber o corpo da mensagem. Se o usuário está num sistema com taxa de conexão baixa, o ganho de não ter que transferir todas as mensagens pode ser perdido no tempo requerido para transferir cada mensagem desejada. Além disso, o mecanismo de filtragem de *spam* pode requerer acesso ao corpo do correio eletrônico. Se buscar o corpo for uma tarefa manual, isso pode tomar mais tempo que receber todas as mensagens, com *spam* ou não. Assim, é necessário criar métodos automatizados que se adeqüem às necessidades dos usuários. Isso poderá ser feito numa segunda etapa, através da modificação dos programas de leitura de correio eletrônico (ver seção 4.1).

O objetivo original deste modelo era racionalizar o transporte de correio eletrônico, evitando o envio de mensagens com conteúdo pesado para a caixa de correio do usuário, o que é desvantajoso principalmente quando a conexão é feita por um enlace de baixa capacidade, como uma linha discada. Assim, ele é ortogonal às outras iniciativas de término de *spam*, e pode ser adotado juntamente com elas.

3.9 CONCLUSÃO

Para resolver ou minorar alguns problemas do correio eletrônico, um novo modelo de transporte (modelo *Pull*) foi proposto. Nele uma mensagem-resumo é criada a partir da mensagem original, contendo informações suficientes (nome do remetente e assunto, entre outras) para que o destinatário decida se quer ou não ver a mensagem original. Caso queira ele clicará em um *link* HTTP na mensagem-resumo e a mensagem original será exibida em seu navegador.

A criação da mensagem resumo é feita no servidor SMTP do remetente e a mensagem original é armazenada em um repositório, que pode ser neste mesmo servidor ou em uma máquina distinta, mas sempre na esfera do remetente. A mensagem original só chegará ao destinatário se este expressamente a requisitar. Caso ele não queira recebê-la ela será apagada do servidor do remetente após um tempo (prazo de validade).

As vantagens desse novo modelo de transporte são: dificultar o *spam*, diminuir o volume de tráfego na *Internet*, possibilitar ao usuário ter uma caixa de entrada distribuída entre diversos servidores, diminuir o volume de mensagens no servidor de destino, diminuir o risco de infecção por vírus, *worms* e cavalos de tróia e a compatibilidade com outras mídias, onde o originador da comunicação é quem arca com os custos.

Como limitações do novo modelo podemos citar: não eliminar completamente a possibilidade do *spam* e introduz um passo extra para que o destinatário veja suas mensagens.

4. IMPLEMENTAÇÃO E TESTES

Para fazer a prova do conceito foi feita uma implementação na linguagem Java, que foi instalada para testes na máquina Itacoatiara do laboratório Midiacom, do Departamento de Engenharia de Telecomunicações da UFF. A versão implementada, desenvolvida para fazer a prova do conceito apenas, é um protótipo funcional, mas que não contém todas as características de uma versão de produção e, portanto, difere um pouco da versão descrita no capítulo anterior, por não estar otimizada para desempenho.

A máquina utilizada foi um Pentium IV 266 GHz, onde já estavam instalados o sistema operacional Linux Slackware 10.1 e o Java versão 1.5.0_06. O servidor WEB utilizado foi o Apache-Tomcat versão 5.5.17.

O programa desenvolvido consiste de sete classes (responsáveis pelo recebimento e armazenamento da mensagem e pela criação e envio da mensagem-resumo ao destinatário) e de um *Servlet*, responsável pela entrega da mensagem original ao destinatário quando este a solicitar.

Para cada mensagem recebida um novo processo (*thread* – um processamento independente, que pode ser realizado simultaneamente a outros) é criado, possibilitando que várias mensagens sejam recebidas simultaneamente. As mensagens são armazenadas em disco utilizando o pacote javax.crypto (criptografia), de forma a impedir que estranhos tenham acesso ao seu conteúdo.

Depois de recebida e armazenada a mensagem original, é criada uma mensagem-resumo (ver Figura 7), que é enviada ao destinatário da mensagem, contendo o seguinte texto:

"Aviso de recebimento de E-mail:

Você recebeu um e-mail de <<*nome do remetente*>> cujo assunto é <<*assunto da mensagem*>>. Para ler essa mensagem, clique no *link* abaixo:"

O *link* que segue é um *link* HTTP para a mensagem original, que está salva no servidor de armazenamento.

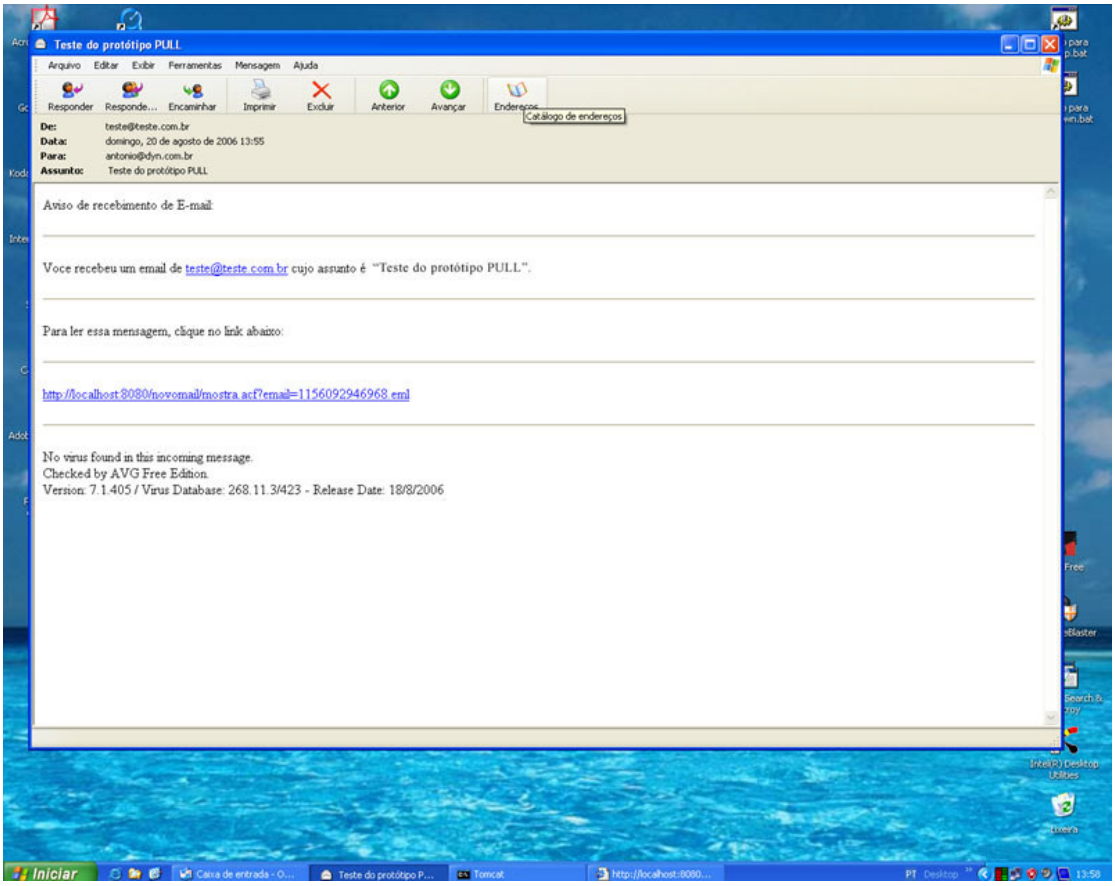

Figura 7: mensagem-resumo enviada o destinatário

A exibição da mensagem original ao destinatário é feita pelo *Servlet*, que recebe a requisição HTTP do *link* da mensagem-resumo e abre uma janela do navegador (ver Figura 8) que estiver instalado na máquina do usuário para mostrar a mensagem. Se houver arquivos anexados eles serão listados nessa tela e o usuário pode abri-los e/ou salvá-los em disco.

Foram realizados testes de aderência aos requisitos e de usabilidade no laboratório Midiacom na UFF e na empresa Dynamis Informática LTDA, onde foram testados os seguintes pontos:

1 – Criação e envio da mensagem-resumo.

2- Fidelidade da mensagem-resumo com a original (se a mensagem-resumo continha o nome correto do remetente, o assunto e a lista de anexos da mensagem original).

3- A inviolabilidade da mensagem original (se o processo de criptografia funcionou corretamente).

4 – A entrega da mensagem original.

Todos os requisitos acima foram executados satisfatoriamente. Testes mais completos (testes de desempenho e carga), que possam simular condições mais realistas de uso, serão efetuados após a construção de um novo protótipo (ver seção 4.1).

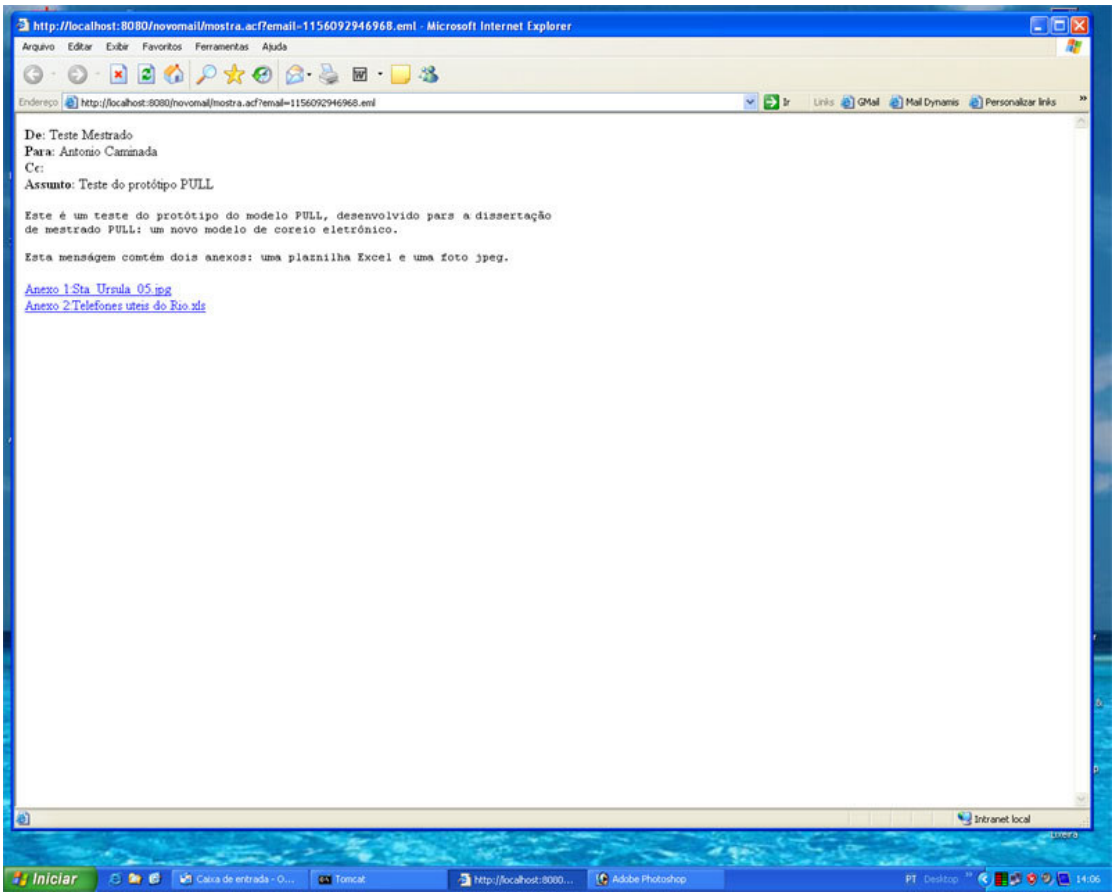

Figura 8: mensagem original exibida no navegador

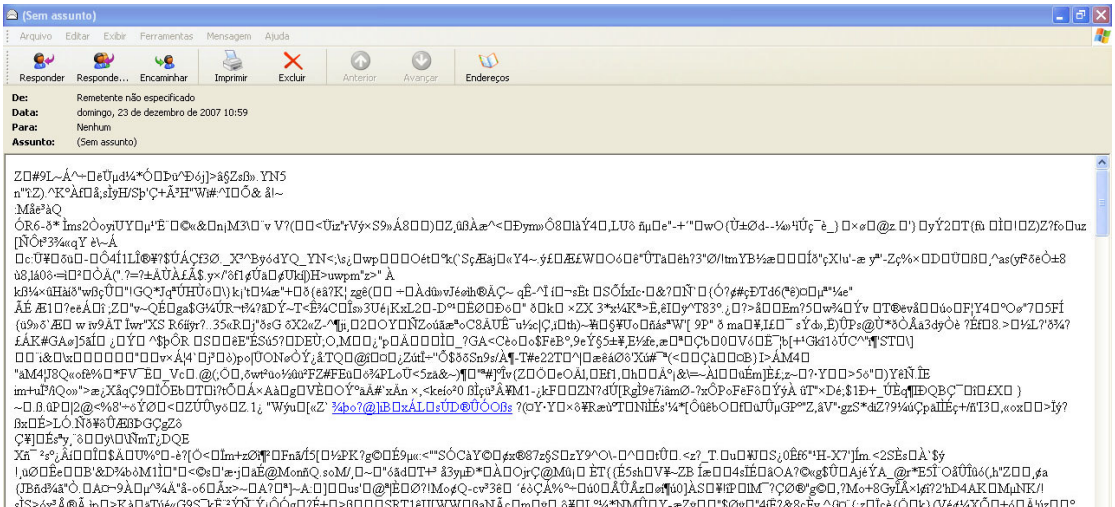

Figura 9: mensagem original criptografada

Na figura 9 é exibida a mensagem original em sua forma criptografada. Isto é o que qualquer pessoa que tentasse ler a mensagem armazenada no repositório veria. Note-se que não é possível identificar o remetente, o assunto nem o corpo da mensagem.

4.1 TRABALHOS FUTUROS

O modelo de correio eletrônico aqui apresentado, apesar de funcionar por si só, pode ser complementado por uma aplicação de cliente (*user agent*), que ofereceria facilidades adicionais, como por exemplo:

- O uso de uma "lista branca", isto é, uma lista de nomes de rementes autorizados, possibilitando ao sistema, sempre que uma mensagem-resumo desse cliente fosse recebida, recuperar automaticamente a mensagem original (o nome do remetente da

mensagem seria comparado com os nomes de uma lista branca criada pelo usuário, e caso estivesse na lista, o UA acionaria o link, recuperando a mensagem original).

- A possibilidade de se realizar a descriptografia no destino, aumentando ainda mais a privacidade dos usuários.

- A possibilidade de se indicar, para cada mensagem, se ela deve ou não ser criptografada.

- A possibilidade de se definir diferentes prazos de validade para as mensagens.

- O encaminhamento de mensagens (suponha que alguém receba uma mensagem-resumo e que esta pessoa deseje encaminhar a mensagem-resumo para que outra pessoa veja a mensagem original).

Com essa aplicação de cliente, poderá também ser aperfeiçoado o sistema de criptografia, utilizando um esquema de chave pública/privada, que poderá funcionar de duas maneiras:

- O remetente enviará, em um novo campo no cabeçalho da mensagem, a chave pública do destinatário. Seu MTA, após criar a mensagem-resumo, criptografará a mensagem original e a armazenará. Quando o destinatário quiser receber a mensagem original, esta será enviada ainda criptografada, e o novo cliente fará a descriptografia de maneira transparente para o usuário.

- Alternativamente, o remetente poderá indicar um repositório onde se encontra a chave pública do destinatário, e o MTA irá nesse repositório, pagará a chave e fará a criptografia da mensagem.

Outra importante melhoria será o desenvolvimento de um protótipo em uma linguagem compilada, como por exemplo C ou C++, para que o programa seja mais eficiente e confiável, já que o protótipo produzido não está otimizado para performance, nem foi submetido a testes de desempenho com cargas elevadas.

4.2 CONCLUSÃO

Para fazer a prova do conceito um protótipo de testes foi desenvolvido e testado (testes de usabilidade) para verificar a praticidade e a reação dos usuários ao novo modelo. A principal vantagem apontada foi a possibilidade de se abrir as mensagens com segurança, sabendo que isso não traria danos à máquina. Como principal desvantagem, foi citado o fato de se ter um passo extra para ver mensagens sabidamente seguras.

Esta dissertação propõe um novo modelo de transporte para o correio eletrônico, que deverá ser complementado com a criação de um *user agent* específico para funcionar nesse modelo. Além disso um protótipo mais robusto deverá ser construído, para possibilitar testes sob condições mais realistas de carga e desempenho.

5. CONCLUSÃO

5.1 O FUTURO DO CORREIO ELETRÔNICO

O futuro do correio eletrônico ainda é incerto. Algumas tendências vêm sendo debatidas, porém, uma coisa é certa: o futuro do correio eletrônico está diretamente relacionado ao *spam*. Ou o se acaba (ou pelo menos, diminui) com o *spam* no correio eletrônico ou o *spam* tornará o correio eletrônico inviável. Um estudo [68] realizado mostra que empresas norte-americanas gastaram U$ 8.900.000.000,00 (oito bilhões e novecentos milhões de dólares) em 2002 com problemas relacionados ao *spam*, sendo 44% em recursos de TI[28], 39% com perda de produtividade de pessoal e 17% com recursos de *help-desk*. Das mais de 30 bilhões de mensagens eletrônicas enviadas diariamente ao redor do mundo, mais de 25% são *spam* [29].

Por causa disso algumas pessoas acham que, num futuro não muito distante, o correio eletrônico será substituído por programas de mensagens instantâneas (*Instant Message*[29] - IM).

Numa pesquisa realizada em 2005 [30], adolescentes se referiram ao correio eletrônico como "algo que é usado para falar com pessoas mais velhas, organizações ou para enviar

[28] Tecnologia da Informação - conjunto de todas as atividades e soluções providas por recursos de computação.

[29] Programas de comunicação em tempo real, onde os participantes estão on-line simultaneamente.

grandes instruções para muitas pessoas". Quando se trata de conversas casuais com seus pares, eles só usam IM.

Não se pode prever o futuro com base apenas nas atitudes de uma geração. É certo que os programas de IM serão cada vez mais populares, mas não substituirão totalmente o correio eletrônico pois têm uma grande desvantagem: a necessidade de uma comunicação síncrona. Uma pessoa pode acordar às 04:00 da manhã e mandar um correio eletrônico para alguém, sabendo que a mensagem será entregue. O mesmo não se pode dizer dos programas de IM, que exigem que os participantes da comunicação estejam *on-line*. Além disso, as mensagens de correio eletrônico podem ser escritas com calma, pensadas e repensadas. As mensagens de IM são instantâneas, e devem ser rapidamente digitadas, caso contrário a conversação se torna impraticável. Para muitas pessoas que têm dificuldade em digitar, isso é muito ruim.

Além disso, o correio eletrônico é usado de várias formas, provavelmente nunca pensadas por seus idealizadores, mas que devido à sua versatilidade acabam sendo possíveis, como por exemplo:

- **Transferência de arquivos**: o correio eletrônico é, com freqüência, utilizado para transferir arquivos, substituindo o FTP, que é um protocolo criado para esse fim.

- **Agenda**: Muitas pessoas usam o correio eletrônico como agenda, enviando mensagens para si mesmas, lembrando de compromissos ou reuniões.

- **Espaço de armazenagem**: com o advento dos correios eletrônicos com capacidade acima de 1GB, muitas pessoas o utilizam como espaço de armazenagem e *backup,* enviando para suas caixas postais arquivos importantes.

Qualquer outra aplicação que porventura viesse a substituir o correio eletrônico teria que oferecer as mesmas facilidades, sob o risco de não ser utilizada e cair no ostracismo.

O cenário mais provável é que o correio eletrônico sobreviva com algumas modificações, para acompanhar mudanças de comportamento que hoje vemos. Cada vez mais reuniões de negócios e acordos comerciais estão se realizando a distância. Video conferências vêm substituindo reuniões, dispensando os participantes de longas e cansativas viagens. Para se inserir neste meio, o correio eletrônico precisa ser visto como uma ferramenta robusta e confiável, para que contratos, por exemplo, possam ser enviados e devolvidos sem o perigo de serem interceptados ou alterados no caminho.

O *spam* vem sendo combatido em diversas frentes e, segundo John Scarrow, gerente geral de anti-*spam* e anti-*phishing* da Microsoft [31], o número de mensagens de *spam* está começando a se estabilizar. É provável que o *spam* evolua para um outro tipo de comunicação comercial, baseado na permissão expressa do usuário [29] que deseje receber comunicações das empresas de seu interesse.

5.2 CONTRIBUIÇÃO

A contribuição do presente trabalho é propor uma nova arquitetura de transporte de correio eletrônico que, sem alterar o SMTP e mantendo a infraestrutura hoje utilizada praticamente inalterada, ofereça soluções para muitos de seus problemas, aumentando a segurança e melhorando a usabilidade, já que os usuários poderão abrir suas mensagens sem medo de estarem causando problemas para si mesmos.

O correio eletrônico é uma das aplicações de maior sucesso da *Internet*. Sua facilidade de uso e versatilidade o transformaram numa ferramenta praticamente indispensável no dia-a-dia. Porém, as mesmas facilidades que o transformaram num sucesso, ameaçam seu futuro. Para que possa continuar existindo, o correio eletrônico deve ser modificado para se adaptar aos novos tempos e a modos de uso que seus idealizadores não tinham em mente quando o criaram.

Essas modificações podem seguir dois rumos: uma aplicação inteiramente nova ou a modificação da existente. Devido à massificação de uso, acreditamos que criar uma aplicação nova, que necessitasse de um protocolo novo ou de uma infraestrutura radicalmente diferente da utilizada atualmente, com ruptura do atual modelo, seria muito difícil de se implementar. Por isso, essa dissertação propõe uma nova forma de utilização do correio eletrônico, mas que preserva seus componentes, permitindo que os dois modelos convivam sem maiores problemas, compartilhando a mesma infraestrutura. Vale salientar que, para os usuários, a mudança não seria complexa: apenas a introdução da mensagem-resumo e a necessidade de acionar um link para receber a mensagem original. Face aos benefícios apresentados, seria um preço pequeno a se pagar.

Com a mudança do modelo *Push* para o modelo *Pull* os usuários poderão escolher quais mensagens querem receber antes que elas possam causar danos às suas máquinas e antes que elas congestionem suas conexões e caixas de entrada. Isso de maneira quase transparente

para eles, que poderão continuar utilizando seus programas de leitura de correio eletrônico e *web-browsers* preferidos da mesma forma que sempre fizeram.

Para chegar ao resultado obtido foi feita uma análise do atual modelo de correio eletrônico: suas origens, sua arquitetura, como funciona e a razão do modelo *Push* ter sido adotado. Após, foi feito um levantamento dos principais problemas que hoje afetam sua usabilidade e que soluções já foram propostas para resolver ou atenuar esses problemas. A seguir, foi proposta a nova arquitetura de transporte, que foi testada com um protótipo construído para esse fim.

O presente trabalho mostra que é possível aproveitar a atual arquitetura de correio eletrônico e, sem alterá-la de maneira substancial, reduzir suas falhas de segurança, possibilitando um uso mais completo da aplicação. Através do protótipo desenvolvido foi constatado que os benefícios da mudança do modelo *Push* pelo *Pull* são grandes – mais segurança, rapidez e confiabilidade, e que as desvantagens são pequenas se comparadas aos benefícios.

A solução aqui proposta é de fácil implementação e resolve ou atenua quase todos os grandes problemas do correio eletrônico, possibilitando que essa aplicação continue sendo usada por muitos anos.

6.REFERÊNCIAS.

[1] Luiz Magalhães, “Correio Eletrônico em Português”, Tese de Mestrado, Departamento de Informática, PUC-Rio 1994.

[2] J. Klensin, “Simple Mail Transfer Protocol“, RFC 2821, abril de 2001.

[3] “The Case For Email Security” - http://luxsci.com/extranet/articles/email-security.html.

[4] “Definition of Spam” - MAPS - http://www.mail-abuse.com/spam_def.html.

[5] “O que são vírus, worms e cavalos de Tróia?” - http://www.microsoft.com/brasil/athome/security/viruses/virus101.mspx.

[6] N. Borenstein, N. Freed, “MIME (Multipurpose Internet Mail Extensions) Part One: Format of Internet Message Bodies “, RFC 2045, novembro de 1996.

[7] Darwin Magazine -http://www.darwinmag.com/read/010102/buzz_mover.html.

[8] “Fidonet” - http://en.wikipedia.org/wiki/Fidonet.

[9] “What is FidoNet” - http://www.fidonet.org/.

[10] “FidoNet: Technology, Use, Tools, and History” - http://www.fidonet.org/inet92_Randy_Bush.txt.

[11] “Usenet” - http://en.wikipedia.org/wiki/UUCP.

[12] Alan David Grier, “A Social History of Bitnet and Listserv”, 1985–1991; 2002 http://www.computer.org/annals/articles/bitnet.htm.

[13] “Webcasting” - http://en.wikipedia.org/wiki/Webcasting.

[14] J. Myers, C. Mellon, M. Rose, “Post Office Protocol - Version 3”, RFC 1939, maio de 1996.

[15] M. Crispin, “Internet Message Access Protocol - Version 4rev1”, RFC 2060, dezembro de 1996.

[16] S. Josefsson, “The Base16, Base32, and Base64 Data Encodings”, RFC 3548, julho de 2003.

[17] Jornal O Globo 17 de junho de 2006.

[18] “Distributed Sender Blackhole List” - http://dsbl.org/main/.

[19] “RFC-Ignorant.org” - http://www.rfc-ignorant.org/.

[20] Microsoft Executive E-Mail: Toward a spam-Free future - http://www.microsoft.com/mscorp/execmail/2003/06-24antispam.asp.

[21] “IIJ Introduces Sender Authentication Technology” - http://www.ip97.com/iij_introduces_sender_authentication_fdc.aspx.

[22] “End of the road for SMTP?” - http://news.com.com/2100-1038_3-5058610.html.

[23] http://www.magma.com.ni/~jorge/im2000/im2000.html.

[24] P. Hoffman, “SMTP Service Extension for Secure SMTP over Transport Layer Security”, RFC 3207, fevereiro de 2002.

[25] Zhenhai Duan, Kartik Gopalan, Yingfei Dong, “Receiver-Driven Extensions to SMTP [I-D]”, IETF Internet Draft. janeiro de 2006.

[26] “IM2000 - electronic mail transport of the future” - http://www.im2000.org/.

[27] “AMTP - Authenticated Mail Transfer Protocol” - http://amtp.bw.org/docs/draft-weinman-amtp-03.txt.

[28] R. Housley, “Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile”, RFC 3280 , abril de 2002.

[29] “The Future of E-Mail” - http://www.clickz.com/showPage.html?page=2168761.

[30] “IM and the future of e-mail” - http://reviews.cnet.com/4520-6028_7-6301361-1.html.

[31] http://www.computerworld.com/action/article.do?command=viewArticleBasic &articleId=111503.

[32] P. Resnick, “Internet Message Format”, RFC 2822 , abril de 2001.

[33] N. Borenstein, N. Freed, “Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types”, RFC 2046, novembro de 1996.

[34] K. Moore, “MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text”, RFC 2047, novembro de 1996.

[35] N. Freed, J. Klensin, J. Postel, “Multipurpose Internet Mail Extensions (MIME) Part Four: Registration Procedures”, RFC 2048, novembro de 1996.

[36] N. Borenstein, N. Freed, “Multipurpose Internet Mail Extensions (MIME) Part Five: Conformance Criteria and Examples”, RFC 2049, novembro de 1996.

[37] R. Gellens, “The Text/Plain Format Parameter”, RFC 2646, agosto de 1999.

[38] S. Hambridge, “Netiquette Guidelines”, RFC 1855, outubro de 1995.

[39] J. Klensin, N. Freed, M. Rose, E. Stefferud, D. Crocker, “SMTP Service Extensions”, RFC 1869, novembro de 1995.

[40] J. Klensin, N. Freed, K. Moore, “SMTP Service Extension for Message Size Declaration”, RFC 1870, novembro de 1995.

[41] G. Vaudreuil, “Enhanced Mail System Status Codes”, RFC 1893, Janeiro de 1996.

[42] P. Hoffman, “SMTP Service Extension for Secure SMTP over TLS”, RFC 2487, janeiro de 1999.

[43] R. Braden, “Requirements for Internet Hosts -- Communication Layers”, RFC 1122, outubro de 1989.

[44] R. Braden, “Requirements for Internet Hosts -- Application and Support”, RFC 1123, outubro de 1989.

[45] J. Linn, “Privacy Enhancement for Internet Electronic Mail: Part I: Message Encryption and Authentication Procedures”, RFC 1421, fevereiro de 1993.

[46] S. Kent, “Privacy Enhancement for Internet Electronic Mail: Part II: Certificate-Based Key Management”, RFC 1422, fevereiro de 1993.

[47] D. Balenson, “Privacy Enhancement for Internet Electronic Mail: Part III: Algorithms, Modes, and Identifiers”, RFC 1423, fevereiro de 1993.

[48] B. Kaliski, “Privacy Enhancement for Internet Electronic Mail: Part IV: Key Certification and Related Services”, RFC 1424, fevereiro de 1993.

[49] Luiz Antonio Pereira, “Métodos de Análise de Sistemas”, Apostila, Curso Análise, Projetos e Gerência de Sistemas – PUC-RJ.

[50] “The Cryptography Introduction and Guide” - http://www.cryptographyworld.com.

[51] http://www.cs.berkeley.edu/~daw/crypto.html.

[52] “Historical Cryptography” - http://starbase.trincoll.edu/~crypto/.

[53] T. J.Kim, “A Downemail System: An Internet Email System Based On Sender Mailbox”, Cheonan National Technical College, Korea, setembro de 2003 - http://ieeexplore.ieee.org/Xplore/login.jsp?url=/iel5/8981/28517/01274317.pdf.

[54] “Como Funciona” - http://email.uol.com.br/antispam/como.jhtm.

[55] D. Turner, K. Ross, “A Comprehensive Architecture for Continuous Media E-Mail on the Internet.”, França, março de 2000 - http://ieeexplore.ieee.org/iel5/93/19845/00917975.pdf?arnumber=917975.

[56] J. Postel, J. Reynolds, “File Transfer Protocol (FTP)”, RFC 959, outubro de 1985.

[57] R. Fielding, UC Irvine, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, “Hypertext Transfer Protocol -- HTTP/1.1”, RFC 2616, junho de 1999.

[58] Dierks, T; Allen, C. (1999) “RFC 2246 - The TLS Protocol”

[59] Housley, R; Polk, W; Ford, W; Solo, D. (2002) “RFC 3280 - Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile”

[60] Ricardo Campanha Carrano, Antonio Caminada, Debora Cristina Muchaluat-Saade, Luiz Claudio Schara Magalhães. TRACE e o Correio Multimídia. In: 25º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2007), Belém. Anais do 25º Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2007)

[61] Duan, Z.; Gopalan, k; Dong, Y. (2005) "DMTP: Controlling Spam Through Message Delivery Differentiation" – http://www.cs.fsu.edu/research/reports/TR-041025.pdf

[62] “Hacker” - http://pt.wikipedia.org/wiki/Hacker

[63] Jonathan B. Postel “Simple Mail Transfer Protocol”, RFC 821, agosto de 1982.

[64] Craig Partridge “Mail Routing and the Domain System”, RFC 974, janeiro de 1986.

[65] David H. Crocker “Standard for the Format of Arpa Internet Text Messages”, RFC 822, agosto de 1982.

[66] J. K. Reynolds “Post Office Protocol”, RFC 918, outubro de 1984.

[67] “Mime” - http://en.wikipedia.org/wiki/MIME

[68] “The State of the Spam Problem” - http://www.educause.edu/ir/library/pdf/erm0357.pdf.

[69] Grady Booch, James Rumbaugh, Ivar Jacobson - Unified Modeling Language User Guide, 2nd Edition, 2005.

[70] “ISC Internet Domain Survey” - http://www.isc.org/index.pl?/ops/ds/

[71] “INTERNET USAGE STATISTICS - The Internet Big Picture” - http://www.internetworldstats.com/stats.htm

[72] “Geolocation by IP Address” - http://www.linuxjournal.com/article/7856

[73] “Trace IP address, email or IM to owner or user” - http://www.abika.com/help/IPaddressmap.htm

[74] “Thomas Bayes” - http://en.wikipedia.org/wiki/Thomas_Bayes

[75] “The DES Algorithm Illustrated” - http://www.aci.net/Kalliste/des.htm

ANEXOS

ANEXO 1: DIAGRAMAS UML DA ARQUITETURA.

Para desenvolver a arquitetura proposta foi escolhida a linguagem Java, da Sun Microsystems, por apresentar facilidade de implementação, ter um grande número de classes prontas e ser multi-plataforma, não necessitando de nenhuma modificação para ser executada em plataformas distintas.

1.1 DESCRIÇÃO UML

Para ilustrar a arquitetura proposta utilizamos a UML – *Unified Modeling Language*, uma linguagem de modelagem de sistemas que se tornou padrão na indústria.

No diagrama de casos de uso serão exibidas as interações do sistema com os atores externos, isto é, as funcionalidades que o sistema oferece para atores (pessoas ou outros sistemas) que estão fora da fronteira do sistema sendo modelado; no diagrama de classes será feita a descrição das classes (na programação orientada a objetos, classes representam entidades relevantes para o sistema sendo modelado. São entidades genéricas que se instanciam na forma de objetos) que compõe o sistema e seus relacionamentos; nos diagramas de transição de estados serão exibidos os possíveis estados para os principais objetos do sistema e, por fim, nos diagramas de seqüência, a lógica de execução dos processos de recebimento e entrega das mensagens.

1.1.1 Diagrama de casos de uso

Os diagramas de casos de uso descrevem um conjunto de cenários de interação entre atores e o sistema. Atores podem ser usuários ou outros sistemas que interagem com o sistema sendo modelado.

– Diagrama de Casos de uso do processo de recebimento das mensagens

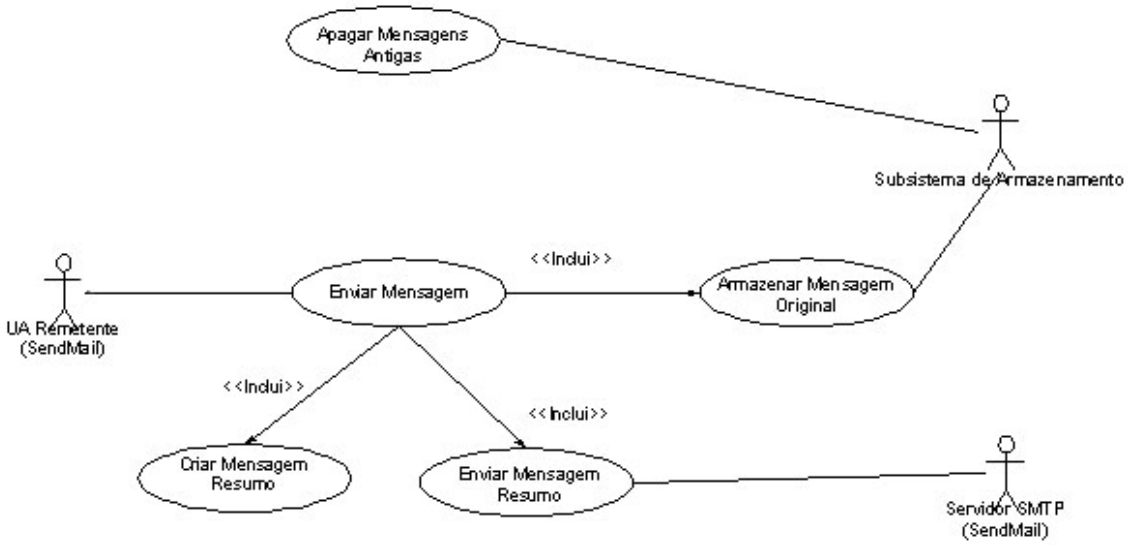

Figura 1: Diagrama de casos de uso do processo de recebimento das mensagens

O diagrama acima representa as interações do processo de recebimento das mensagens com os atores externos (pessoas ou sistemas). Esses atores são o *user agent* do remetente, um subsistema de armazenamento (que pode ser o *file system object* do sistema operacional) e o servidor SMTP. Cada ator pode interagir em um ou mais casos de uso. Alguns casos de uso incluem outros casos de uso, isto é, ao executar o caso de uso principal, necessariamente serão executados os casos de uso incluídos, como por exemplo, sempre que se enviar uma mensagem, terá que ser feita a criação da mensagem-resumo, o envio da mensagem-resumo e o armazenamento da mensagem original. A direção das setas indica qual o caso de uso que inclui (de onde parte a seta) e quais os caso de uso incluídos (aonde chegam as setas). A leitura do gráfico deve ser feita em conjunto com as descrições (abaixo) que explicitam, passo-a-passo, todas as etapas dos casos de uso. Cada caso de uso deve ter um curso típico e pode ter um ou mais cursos alternativos. O curso típico representa a interação padrão, isto é, o que deve acontecer em uma situação normal de uso. Os cursos alternativos são desvios que ocorrem se algo de anormal acontecer, como por exemplo, no caso de uso Enviar Mensagem, o curso típico indica que o sistema, ao receber a mensagem original na porta 25, deve criar a mensagem-resumo, armazenar a mensagem original e por fim, enviar a mensagem resumo ao destinatário. Caso o sistema, não consiga criar a mensagem-resumo ou não consiga armazenar

a mensagem original (supondo que não haja espaço em disco, por exemplo), ele deverá executar o curso alternativo, que diz que a mensagem original deverá ser enviada ao destinatário.

Caso de Uso:	Enviar Mensagem
Ator:	UA do remetente - SendMail
Descrição:	Recebe e processa a mensagem de correio eletrônico enviada pelo UA do remetente
Pré-Condições:	Conexão TCP estabelecida na porta 25, usuário autenticado
Pós-Condições:	Uma nova mensagem criada apenas com o cabeçalho da mensagem original

CURSO TÍPICO

1. Sistema recebe a mensagem do protocolo TCP na porta 25.
2. Executa caso de uso Criar Mensagem Resumo.
3. Executa caso de uso Armazenar Mensagem Original.
4. Executa caso de uso Enviar Mensagem Resumo.

Fim do caso de uso.

CURSO ALTERNATIVO	
Curso Alternativo 01	
Descrição:	No Passo 02 ou 03 do Curso Típico houve algum erro.

1. Encaminha a mensagem original ao servidor SMTP para que esta seja enviada ao destinatário.

Fim do caso de uso.

Caso de Uso:	Criar Mensagem Resumo
Descrição:	Cria uma nova mensagem (mensagem-resumo) com o cabeçalho da mensagem original e um *link* HTTP para a mensagem original, que será armazenada no disco
Pré-Condições:	Uma mensagem recebida
Pós-Condições:	Uma nova mensagem criada a partir da mensagem original

CURSO TÍPICO

1. Analisa a mensagem original, identificando o cabeçalho e o corpo.
2. Copia o cabeçalho.
3. Cria uma nova mensagem com o cabeçalho recém copiado.
4. Adiciona ao corpo da nova mensagem um *link* HTTP para a mensagem original que foi salva.

Fim do caso de uso.

Caso de Uso:	Enviar Mensagem Resumo
Ator:	Servidor SMTP - SendMail
Descrição:	Envia a mensagem-resumo para o servidor SMTP para que este a encaminhe ao destinatário.
Pré-Condições:	Uma conexão TCP estabelecida com o servidor SMTP
Pós-Condições:	Mensagem enviada

CURSO TÍPICO

1. Envia a mensagem ao servidor SMTP para que este a encaminhe para o destinatário.

Fim do caso de uso.

CURSO ALTERNATIVO	
Curso Alternativo 01	
Descrição:	No Passo 01 do Curso Típico a mensagem não pôde ser enviada.

1. Envia a mensagem de erro ao servidor SMTP para que este a encaminhe para o destinatário.

Fim do caso de uso.

Caso de Uso:	Armazenar Mensagem Original
Ator:	Subsistema de Armazenamento
Descrição:	Armazena em disco a mensagem original
Pré-Condições:	Mensagem original recebida
Pós-Condições:	Mensagem armazenada no disco

CURSO TÍPICO

1. Gera um identificador aleatório que será o nome do arquivo da mensagem original.
2. Armazena a mensagem original no disco rígido com o nome gerado no passo 1.

Fim do caso de uso.

Caso de Uso:	Apagar Mensagens Antigas
Ator:	Subsistema de Armazenamento
Pré-Condições:	-
Pós-Condições:	-

CURSO TÍPICO

1. Diariamente o sistema percorre as pastas de armazenagem de mensagens em busca de mensagens cujo "*timestamp*" esteja expirado.
2. Ao encontrar mensagens na situação acima pede ao subsistema de armazenamento que as apague do disco.

Fim do caso de uso.

- Diagrama de casos de uso do processo de envio da mensagem solicitada

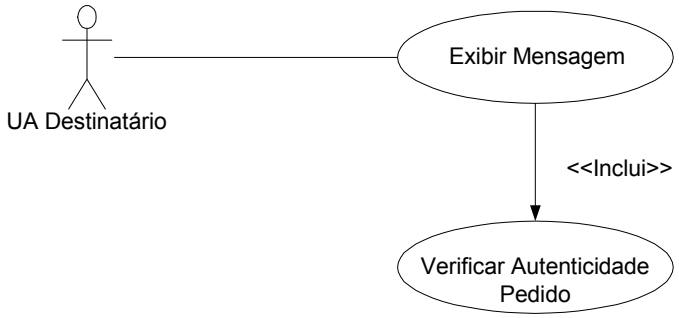

Figura 2: Diagrama de casos de uso do processo de envio da mensagem solicitada

Caso de Uso:	Exibir mensagem
Ator:	UA destinatário
Descrição:	Entrega a mensagem pedida
Pré-Condições:	Um pedido de entrega de uma mensagem
Pós-Condições:	Mensagem entregue

CURSO TÍPICO

1. Executa caso de uso Verificar Autenticidade Pedido.
2. Localiza a mensagem pedida.
3. Exibe a mensagem.

Fim do caso de uso

CURSO ALTERNATIVO	
Curso Alternativo 01	
Descrição:	No Passo 01 do Curso Típico ocorre falha na verificação de segurança

1. Retorna ao usuário uma mensagem informando o erro e pede que tente novamente.

Fim do caso de uso.

Caso de Uso:	Verificar Autenticidade Pedido
Descrição:	Verifica se a pessoa que faz o pedido tem permissão para receber a mensagem
Pré-Condições:	-
Pós-Condições:	-

CURSO TÍPICO

1. Executa a verificação de segurança.

Fim do caso de uso.

1.1.2 Diagrama de Classes

O diagrama de classes é de fundamental importância na programação orientada a objetos. Ele mostra as classes, suas relações (herança, agregação, associação, etc), seus métodos e atributos. As classes definem que tipos de objetos serão criados (um objeto é uma instância de uma classe). Cada classe pode ter métodos (ações que os objetos dessa classe poderão executar) e atributos (características que os objetos da classe deverão ter). As linhas entre as classes indicam qual o relacionamento entre elas, como por exemplo, uma mensagem pode ter nenhum ou muitos anexos (0..*), mas um anexo só pode pertencer a uma única mensagem (1); cada mensagem original, só pode ter sido enviada por um remetente, mas um remetente pode enviar várias mensagens originais (*). O diagrama de classes é acompanhado pela descrição dos métodos e atributos, que explica o que eles representam.

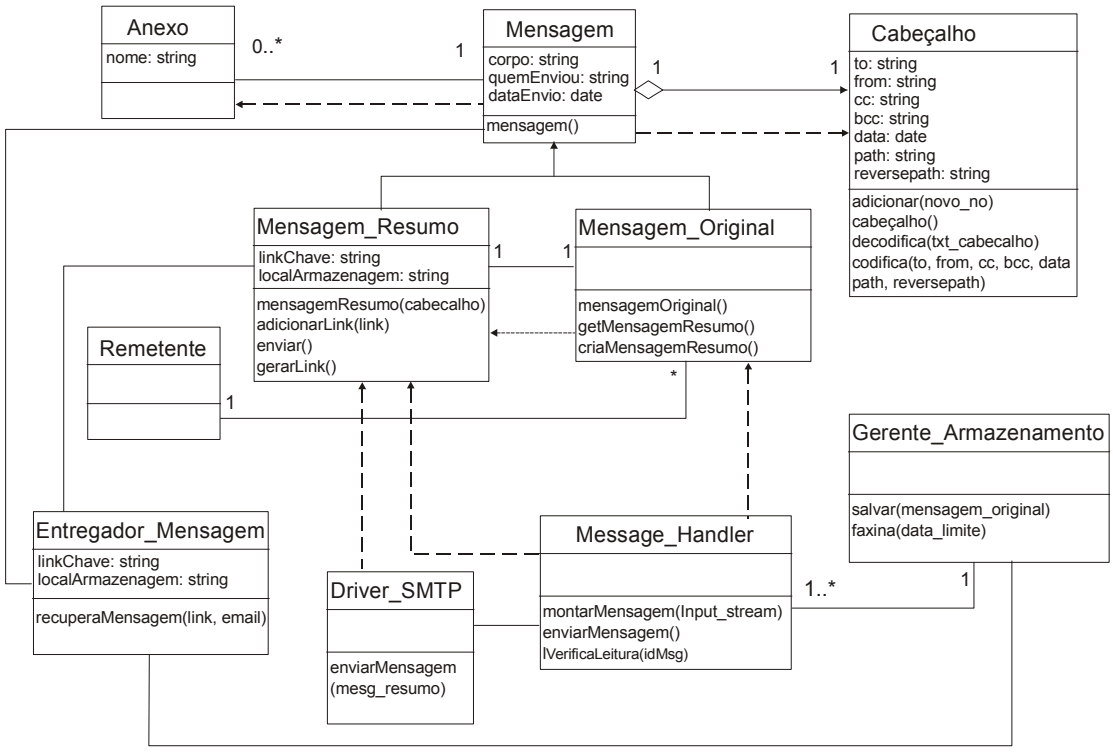

Figura 3: Diagrama de Classes

Diagrama de Classes – Descrição de métodos e atributos

1 - Classe Mensagem (superclasse):

1A – Atributos:

corpo: String – corpo da mensagem recebida.

quemEnviou: String – o remetente da mensagem.

dataEnvio: date – data de envio da mensagem.

1B – Métodos:

mensagem(): construtor - recebe o input *stream* do *socket* e monta a mensagem de correio eletrônico (mensagem original).

2 – Classe Mensagem_Resumo:

2A – Atributos:

*link*Chave: String – *link* que contém a chave para a recuperação da mensagem original.

localArmazenagem: String – local (caminho) onde o arquivo contendo a mensagem original será armazenado.

2B – Métodos:

mensagemResumo(cabeçalho): construtor – cria a mensagem-resumo a partir do cabeçalho recebido como parâmetro.

adicionar*Link*(*link*): adiciona à mensagem-resumo o *link* com a chave que possibilitará a recuperação da mensagem original.

enviar(): entrega a mensagem ao *driver* SMTP para que esse a envie ao destinatário.

gerar*Link*(): produz o *link* http que será usado na recuperação da mensagem.

3 – Classe Mensagem_Original:

3A – Métodos:

mensagemOriginal(): construtor – monta a mensagem original a partir de um inputStream.

getMensagemResumo(): recupera a mensagem-resumo.

criaMensagemResumo(): cria a mensagem-resumo.

4 – Classe Anexo:

4A – Atributos:

nome: String – o nome do arquivo anexado.

5 – Classe Cabeçalho:

5A – Atributos:

to: String – nome do destinatário da mensagem.

from: String – nome do destinatário da mensagem.

cc: String – nome dos destinatários que receberão uma cópia da mensagem.

bcc: String – nome dos destinatários que receberão uma cópia oculta da mensagem.

data: date – data de envio da mensagem.

path: String – caminho percorrido pela mensagem (nós intermediários).

reversePath: String – caminho percorrido pela mensagem em ordem reversa.

5B – Métodos:

adicionar(novo_no): adiciona um novo nó ao cabeçalho.

cabeçalho(): construtor – instancia um objeto cabeçalho, adicionando os campos necessários.

decodifica(txt_cabeçalho): separa os campos do cabeçalho a partir do cabeçalho completo.

codifica(to, from, cc, bcc, data, path, reversePath): monta o cabeçalho completo a partir dos campos separados.

6 – Classe Gerente_Armazenamento:

6A – Métodos

salvar(mensagem_original): armazena em disco a mensagem original.

faxina(data_limite): apaga do disco as mensagens cuja data de envio seja superior à data limite recebida como parâmetro.

7 – Classe Message_Handler:

7A – Métodos:

montarMensagem(input_stream): recebe um *input stream* e monta a mensagem original.

enviarMensagem(): envia a mensagem-resumo.

VerificaLeitura(idMsg): verifica se a mensagem já foi lida por todos os destinatários (caso seja endereçada a mais de um) ou se já foi lida pelo destinatário (caso seja endereçada a somente um).

8 – Classe Entregador_Mensagem:

8A - Atributos:

*link*Chave: String – *link* que contém a chave para a recuperação da mensagem original.

localArmazenagem: String – local (caminho) onde o arquivo contendo a mensagem original será armazenado.

8B – Métodos:

recuperaMensagem(*link*, email): entrega, quando solicitada, a mensagem original.

9 – Classe Driver_SMTP:

9A – Métodos:

enviarMensagem(msg_resumo): envia a mensagem-resumo.

1.1.3 Diagramas de Transição de Estados

Os diagramas de transição de estado são usados para mostrar o comportamento do sistema. Eles descrevem os possíveis estados (um estado é uma condição durante a vida de um objeto na qual ele efetua uma ação, satisfaz uma condição ou espera por um evento) [49] pelos quais um objeto pode passar à medida que os eventos vão acontecendo. A bola cheia indica o ponto de partida, isto é, o estado inicial. A bola vazada indica o estado final. As setas indicam o que deve acontecer para que haja mudanças de estado. No diagrama abaixo, por exemplo, o estado inicial é "mensagem em construção". O objeto mensagem permanecerá neste estado enquanto estiver recebendo o *input stream* da porta 25 e, enquanto estiver neste estado, ele executará a criação da mensagem-resumo e o salvamento da mensagem original. Somente quando esses passos tiverem sido executados e a mensagem-resumo estiver pronta o estado será alterado para "mensagem sendo enviada".

– Diagrama de Estados do Objeto Mensagem

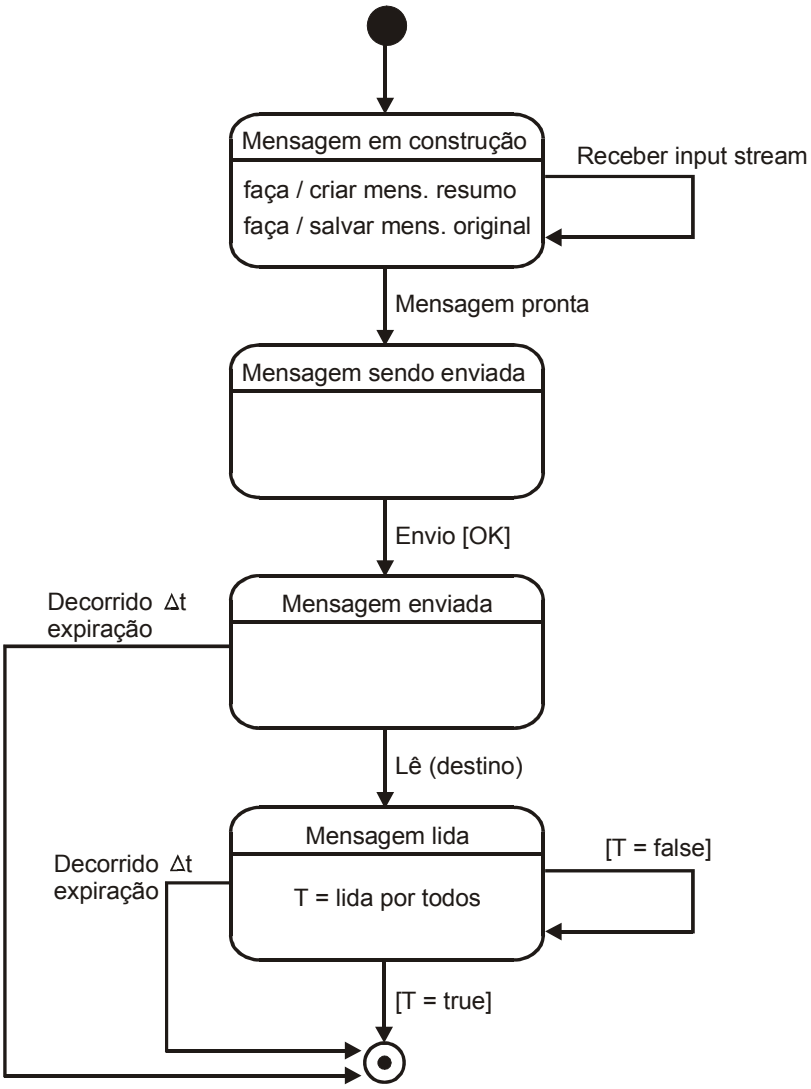

Figura 4: Diagrama de Estados do Objeto Mensagem

– Diagrama de Estados do Objeto Message Handler

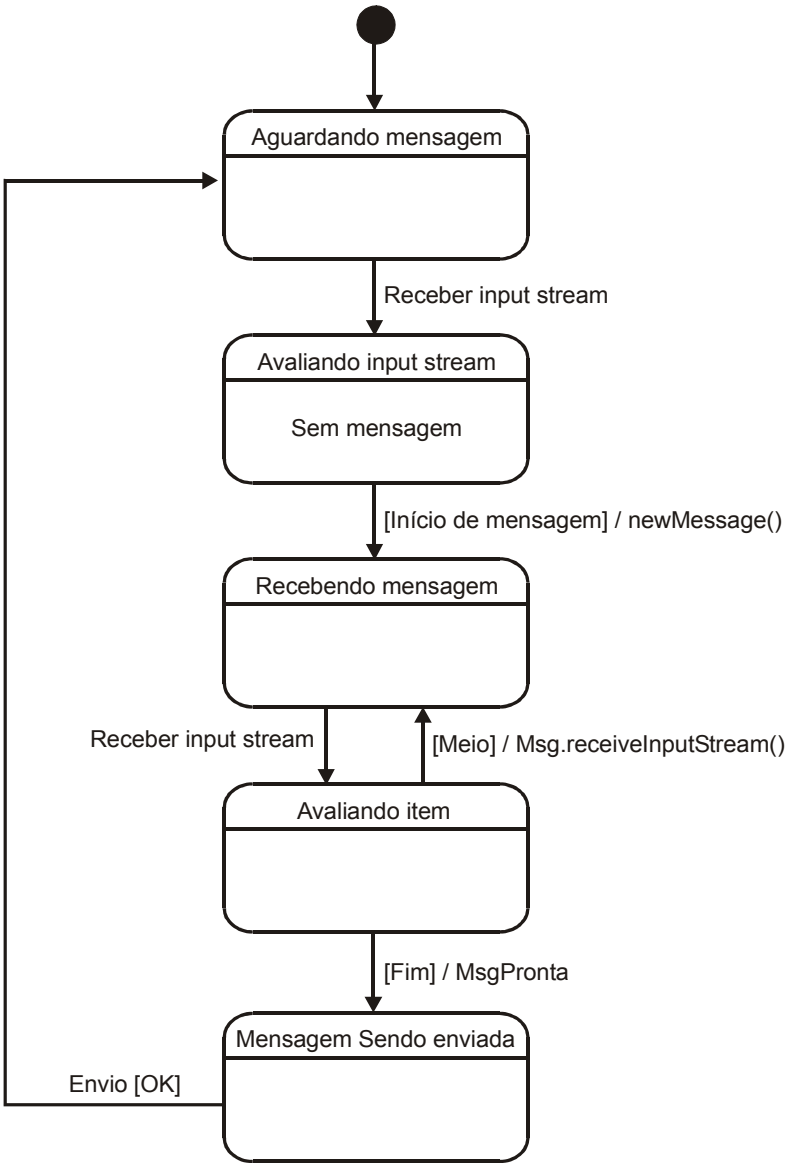

Figura 5: Diagrama de Estados do Objeto Message Handler

1.1.4 Diagramas de Seqüência

Os diagramas de seqüência descrevem o funcionamento do sistema (como grupos de objetos colaboram em algum comportamento do sistema) de maneira visual, permitindo a validação da lógica de execução do programa. Tipicamente um diagrama de seqüência captura o comportamento de um único cenário de um caso de uso [49]. No diagrama de seqüência as caixas representam objetos (a ordem em que eles aparecem é irrelevante), as linhas verticais (linhas de vida) a passagem do tempo, o bonequinho representa um ator que interage com o cenário representado e as setas horizontais indicam as ações (métodos) executadas pelos objetos. No diagrama abaixo, por exemplo, o ator (UA remetente) inicia uma ação (EnviarMensagem()), ativando o objeto MessageHandler. O objeto MessageHandler, por sua vez, executa a ação (MontarMensagem()) sobre si mesmo e, após tê-la executado, instancia um objeto MensagemOriginal. A leitura é feita na direção das setas e de cima para baixo.

– Diagrama de seqüência do processo de recebimento das mensagens

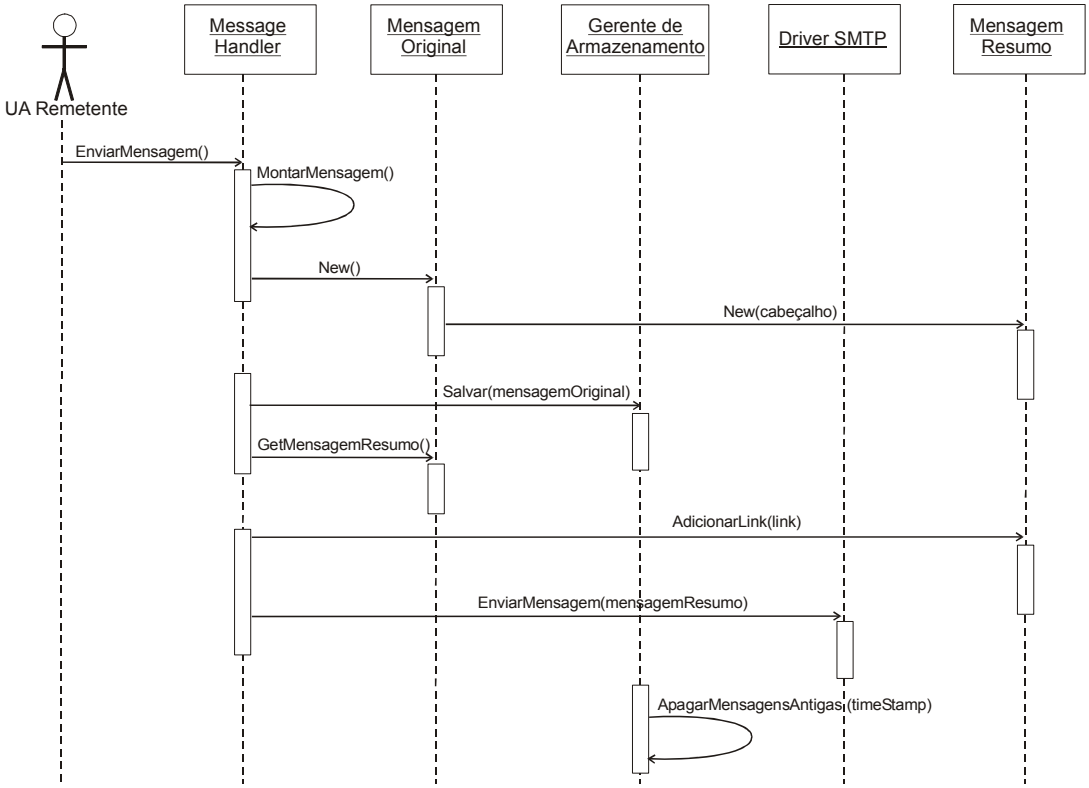

Figura 6: Diagrama de seqüência do processo de recebimento das mensagens

– Diagrama de seqüência do processo de entrega das mensagens

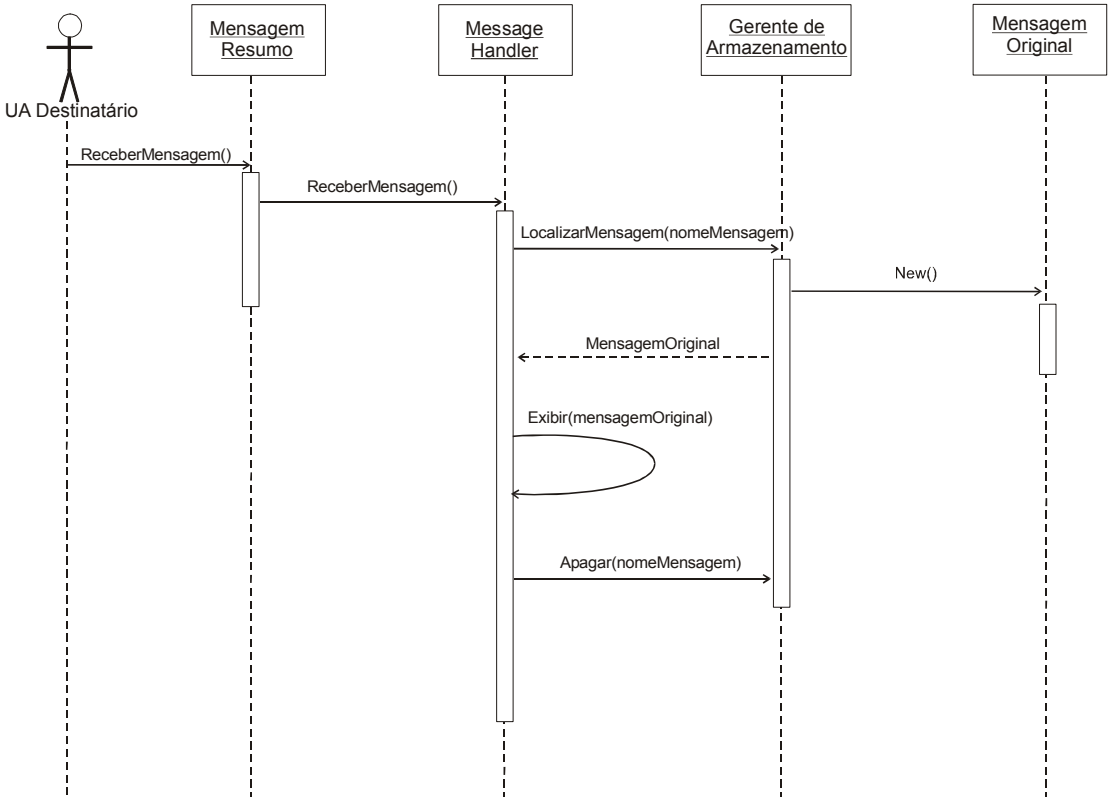

Figura 7: Diagrama de seqüência do processo de entrega das mensagens

1.2 CONCLUSÃO

A UML tornou-se a linguagem padrão para a modelagem de sistemas informatizados. Nesse capítulo apresentamos os principais diagramas UML para exemplificar como será a nova arquitetura de transporte.

No diagrama de casos de uso foram exibidas as interações do sistema com os atores externos; no diagrama de classes foi feita a descrição das classes que compõe o sistema e seus relacionamentos; nos diagramas de transição de estados foram exibidos os possíveis estados para os principais objetos do sistema e, por fim, nos diagramas de seqüência, a lógica de execução dos processos de recebimento e entrega das mensagens.

ANEXO 2: PROCEDIMENTO PARA INSTALAÇÃO DO PROTÓTIPO IMPLEMENTADO

Baixar e instalar o java SDK 5 (mais completo que a JVM)

2. Baixar e instalar o java activation framework

3. Baixar e instalar o java mail API

4. Baixar e instalar o Tomcat 5.5 (apache.org)

5. Cria um diretório para a aplicação (ex: /usr/local/filterserver/)

6. Copiar o jar do serviço para esta pasta

7. Modificar o CLASSPATH para incluir o activation framework, o java mail API e jar do projeto.

8. Criar uma pasta para a aplicação WEB no Tomcat (ex: CATALINA_HOME\webapps\filterserver)

9. Copiar a parte WBE para a pasta acima (todos os arquivo e sub-diretórios).

10.Modificar o arquivo server.xml em CATALINA_HOME\conf para incluir a linha abaixo:

<Context path="/filterserver" docBase="filterserver" debug="0" reloadable="true" crossContext="true" /> dentro da sessão [<Host name="localhost" debug="0" appBase="webapps" unpackWARs="true" autoDeploy="true">] logo abaixo do comentário [<!-- Tomcat Root Context -->]

11. Reiniciar o Tomcat

ANEXO 3: CÓDIGO FONTE DO PROTÓTIPO IMPLEMENTADO

Classe FilterMail – classe inicial

```
package acfos;
import java.util.Properties;

/**
 * This class starts the server to initiate the filtering routine
 */
public class FilterMail {

	private static int port = 0;
	private FilteringSmtpServer filter;
	private static boolean debug = false;
	private static Properties props;

	public static void main(String[] args)
	{
		Param par = new Param();
		props = par.getProperties();

		port = Integer.parseInt(props.getProperty("in_por"));
		//System.out.println("Porta: " + port);

		// assures that a valid port will be chosen. If no port is
		// specified, 3000 is designated as the server port.
		if(args.length>0){
			try {
				port = Integer.parseInt(args[0]);
			}
			catch (Exception e) {
				System.out.println("Erro no parametro: " + e.getMessage());
			}
		}

		//Starts the server thread.
		new FilteringSmtpServer(port,debug).start();

		System.out.println("Starting mail filter server on port " + port);

	}
}
```

Classe FilteringSmtpServer – cria um socket para receber as mensagens e abre uma thread para cada uma delas.

```
package acfos;
```

```
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

/**
 * This is the main server class, it plugs a serversocket to the assigned
 * port and receives the connections, starting different threads to treat
 * each connection.
 */

public class FilteringSmtpServer extends Thread{

	private Socket socket;
	private ServerSocket server;
	boolean debug;

	public FilteringSmtpServer(int port,boolean debug)
	{
		this.debug = debug;
		try {
			server = new ServerSocket(port, 30000);
			if(debug) System.out.println("Listening for connections on port "+
server.getLocalPort());

		} catch (IOException e) {
			System.out.println("Error starting Serversocket.");
			e.printStackTrace();
		}
	}

	public void run()
	{
		while (true) {
			Socket incoming;
			try {
				// accepts the connection
				incoming = server.accept();
				if(debug) System.out.println("Connection established with "+
incoming);
				// starts the filtering thread
				new MessageHandler(incoming, debug).start();

			} catch (IOException e) {
				System.out.println("Error accepting connection.");
				e.printStackTrace();
			}
		}
	}
}
```

Classe MessageHndler – recebe o inputStream do socket, monta a mensagem e faz a criptografia.

```
package acfos;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.PrintWriter;
import java.io.Writer;
import java.net.Socket;
import java.util.Properties;
import java.util.Vector;

//import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.mail.Internet.AddressException;
import javax.mail.Internet.InternetAddress;

/**
 * This class receives the socket and filters the message.
 */
public class MessageHandler extends Thread {

private Socket socket;
private InputStream inputStream = null;
private Writer writer = null;
private BufferedReader bufferedReader = null;
private PrintWriter printWriter = null;
private boolean sessionOver = false;
private boolean debug;
private String line;
private InternetAddress from = null;
private InternetAddress to = null;
private boolean hasData = true;
private MessageSaver saver;
private SmtpServer sender;
private String messageAddress;
private String assunto;
private String path;

        // a recipient for the header
        private Vector messageHeader;
```

```
        // a recipient for the body
        private Vector messageBody;

    // Helper constants
    private static final String MAIL_FROM="MAIL FROM:";
    private static final String RCPT_TO="RCPT TO:";
    private static final String DATA="DATA";
    private static final String OK="250 OK";
    private static final String BANNER="220 Hello from the Filter Server";
    private static final String HELO="HELO";
    private static final String QUIT="QUIT";
    private static final String SUBJECT="Subject:";
    private static final String END_DATA=".";

    // constructor, assigns received parameters
        public MessageHandler(Socket socket, boolean debug)
        {
                this.socket = socket;
                this.debug = debug;
                Param par = new Param();
                Properties props = par.getProperties();
                path = props.getProperty("msg_path");
        }

        public void run()
        {
                try {
                        messageHeader = new Vector();
                        messageBody = new Vector();
                        saver = new MessageSaver();
                        sender = new SmtpServer();
                        inputStream = socket.getInputStream();
                        bufferedReader = new BufferedReader(new
InputStreamReader(inputStream));
                        printWriter = new PrintWriter(socket.getOutputStream());
                        printWriter.println(BANNER);
        printWriter.flush();

        // while the protocol has not been completed, do.
                        while(!sessionOver)
                        {
                                // read a new line.
                                line = bufferedReader.readLine();

                                // The IFs below conform to the SMTP protocol and fulfill
                                // the handshake to receive the message.
                                if(line.startsWith(HELO)) {
                messageHeader.add(line);
                printWriter.println(OK);
                printWriter.flush();
```

```
            if(debug) System.out.println(line+" :-: "+OK);
        }
                        else if(line.startsWith(MAIL_FROM)) {
            String fromString = line.substring(MAIL_FROM.length(), line.length());
            from = new InternetAddress(fromString);
            messageHeader.add(line);
            printWriter.println(OK);
            printWriter.flush();
            if(debug) System.out.println(line+" :-: "+OK);
        }
                        else if(line.startsWith(RCPT_TO)) {
            String toString = line.substring(RCPT_TO.length(), line.length());
            to = new InternetAddress(toString);
            messageHeader.add(line);
            printWriter.println(OK);
            printWriter.flush();
            if(debug) System.out.println(line+" :-: "+OK);
        }
                        else if(line.startsWith(DATA)) {
                                printWriter.println("354 Ready.");
                                printWriter.flush();

                                // After the header has been received, receives the
                                // data until it receives a "." alone(by testing
END_DATA).
                                while(hasData)
                                {
                                        line = bufferedReader.readLine();
                                        if(debug) System.out.println(line);
                                        messageBody.add(line);
                                        if(line.startsWith(SUBJECT)){
                                                assunto =
line.substring(SUBJECT.length(), line.length());
                                        }
                                        if(line.equals(END_DATA))
                                                hasData = false;
                                }
                                printWriter.println(OK);
                                printWriter.flush();
                        }
                        // finishes the communication with the email client sending
QUIT.
                        else if(line.startsWith(QUIT)) {
            sessionOver=true;
            printWriter.println(OK);
            printWriter.flush();
        }
                }
            } catch (IOException e) {
                    e.printStackTrace();
```

```
                } catch (AddressException e) {
                        System.out.println("Invalid address on email header");
                        e.printStackTrace();
                }
                // Stores the message in a EML file and gets the address.
                messageAddress = saver.saveInEmlFile(messageHeader,messageBody);

                //Criprtografia
                //Outro código (caso esse não funcione) http://www.oop-
reserch.com/mercury_1_0.html
                try {
              // Generate a temporary key. In practice, you would save this key.
              // See also e464 Encrypting with DES Using a Pass Phrase.
                        //http://javaalmanac.com/egs/javax.crypto/DesFile.html
              //SecretKey key = KeyGenerator.getInstance("DES").generateKey();

              //saveKey(key);
                        SecretKey key = loadKey();
              // Create encrypter/decrypter class
              DesEncrypter encrypter = new DesEncrypter(key);
              FileInputStream fileIn;
              fileIn = new FileInputStream(path + messageAddress + ".eml");
              // Encrypt
              encrypter.encrypt(fileIn,
                 new FileOutputStream(path + messageAddress + ".eml.crp"));      //.crp
              fileIn.close();

              //Apaga o arquivo original
              try {
                        boolean success = (new File(path + messageAddress + ".eml")).delete();
                        if (!success) {
                           // Deletion failed
                           System.err.println("Failed to delete " + path + messageAddress +
".eml");
                         }
              }catch (SecurityException e) {
                           System.err.println("Unable to delete " + path + messageAddress +
".eml" + "("
                              + e.getMessage() + ")");
                     }

              // Decrypt
              //encrypter.decrypt(new FileInputStream(path + messageAddress + ".eml.crp"),
              //    new FileOutputStream(path + messageAddress + "_decripto.eml"));

                 if(messageAddress!=null)
                        sender.sendMessage(from, to, assunto, messageHeader,
messageAddress, key.toString());
            }
                catch (Exception e) {
```

```
            }
                //Fim criptografia
        }
        public void saveKey(SecretKey chave) {
                try {
                        FileOutputStream out = new FileOutputStream(path + "base.key");
                        ObjectOutputStream s = new ObjectOutputStream(out);
                        s.writeObject(chave);
                        s.flush();
                } catch (FileNotFoundException e) {
                        System.err.println("Erro ao salvar a chave " + e.getMessage());
                } catch (IOException e) {
                        System.err.println("Erro ao salvar a chave " + e.getMessage());
                }
        }
        public SecretKey loadKey() {
                SecretKey chave = null;
                try {
                        FileInputStream in = new FileInputStream(path + "base.key");
                        ObjectInputStream s = new ObjectInputStream(in);
                        chave = (SecretKey) s.readObject();
                } catch (ClassNotFoundException e) {
                        System.err.println("Erro ao salvar a chave " + e.getMessage());
                } catch (FileNotFoundException e) {
                        System.err.println("Erro ao salvar a chave " + e.getMessage());
                } catch (IOException e) {
                        System.err.println("Erro ao salvar a chave " + e.getMessage());
                }
                return chave;
        }
}
```

Classe MessageSaver – salva a mensagem original (já criptografada) no disco.

```
package acfos;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Date;
import java.util.Properties;
import java.util.Vector;

/**
 * This class saves the message. It can be implemented to save it
 * to disc in any format ot to a database
 */

public class MessageSaver {
```

```
	private BufferedWriter log = null;
	private Date data;
	private Long endereco;
	private String path;

	/**
	 *	Saves the message in a <i>.EML</i> file, according to a name given by the
	 *  current date given in milliseconds.
	 *
	 * @param header a Vector containing the header.
	 * @param body a Vector containing the data.
	 * @return the name of the EML file
	 */
	public String saveInEmlFile(Vector header, Vector body)
	{
		Param par = new Param();
		Properties props = par.getProperties();
		path = props.getProperty("msg_path");

		data = new Date();
		try {
			endereco = new Long(data.getTime());
			log = new BufferedWriter(new FileWriter(path + endereco + ".eml"));

			for(int i=0;i<header.size();i++)
			{
				log.write((String)header.elementAt(i) + "\r\n");
				log.flush();
			}
			for(int i=0;i<body.size();i++)
			{
				log.write((String)body.elementAt(i) + "\r\n");
				log.flush();
			}
			log.close();
			return endereco.toString();

		} catch (IOException e) {
			System.out.println("Error writing to file");
			e.printStackTrace();
			return null;
		}
	}
}
```

Classe SmtpServer – envia a mensagem-resumo para o destinatário.

```
package acfos;
```

```
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.util.Date;
import java.util.Properties;
import java.util.Vector;

import javax.mail.Address;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Multipart;
import javax.mail.Session;
import javax.mail.Transport;
import javax.mail.Internet.InternetAddress;
import javax.mail.Internet.MimeBodyPart;
import javax.mail.Internet.MimeMessage;
import javax.mail.Internet.MimeMultipart;

/**
 * This class send the filtered message to the receiver through Smtp,
 * along the address to find the complete message
 */
public class SmtpServer {

	private DataOutputStream os = null;
	private DataInputStream is = null;
	private Address[] address;
	private boolean debug = true;
	private Properties propriedade;
	private String web;

	/**
	 * Sends a message to a SMTP server.
	 *
	 * @param originator the email address of the sender
	 * @param recipient the email address of the receiver
	 * @param messageHeader the vector containing the header of the message
	 * @param file the name of the file this message is stored
	 */
	public void sendMessage(InternetAddress originator, InternetAddress recipient, String
subject, Vector messageHeader, String file, String key)
	{
		Param par = new Param();
		propriedade = par.getProperties();

		String host = propriedade.getProperty("smtp");
		web = propriedade.getProperty("web");
		String portaSaida = propriedade.getProperty("out_Port");

		/** UFF
```

```
                String msgText1 = "Voce recebeu um email de "+originator.toString()+" que
foi filtrado";
                String msgText2 = "Para receber o corpo desta mensagem clique no link
abaixo";
                String msgText3 =
"http://www.midiacom.uff.br/~antonio/messages/nome="+file+".eml";

                String to = recipient.toString();
                String from = originator.toString();
//              String host = "smtp.dyn.com.br";
                String host = "icarai.midiacom.uff.br";
//               create some properties and get the default Session
                Properties props = new Properties();
                props.put("mail.smtp.host", host);
                Session session = Session.getDefaultInstance(props, null);
                */

//              CASA
                String msgText0 = "Aviso de recebimento de E-mail:";
                String msgText1 = "Voce recebeu um email de "+originator.toString()+" cujo
assunto é '"+subject+"'.";
                String msgText2 = "Para ler essa mensagem, clique no link abaixo:";
                String msgText3 = web + "?email=" + file + ".eml"; //.crp

                String to = recipient.toString();
                String from = originator.toString();

//               create some properties and get the default Session
                Properties props = new Properties();
                props.put("mail.smtp.host", host);
                Session session = Session.getDefaultInstance(props, null);
                //Configuração da porta de envio de SMTP (mudar para parâmetro no arquivo
de conf)
                props.put("mail.smtp.port", portaSaida);

                session.setDebug(debug);
                try {
//                              create a message
                                MimeMessage msg = new MimeMessage(session);
                                msg.setFrom(new InternetAddress(from));
                                InternetAddress[] address = {new InternetAddress(to)};
                                msg.setRecipients(Message.RecipientType.TO, address);
                                msg.setSubject(subject);
                                msg.setSentDate(new Date());
//                              create and fill the first message part
                                MimeBodyPart mbp0 = new MimeBodyPart();
                                mbp0.setText(msgText0);
                                MimeBodyPart mbp1 = new MimeBodyPart();
                                mbp1.setText(msgText1);
                                MimeBodyPart mbp2 = new MimeBodyPart();
```

```
                                mbp2.setText(msgText2);
                                MimeBodyPart mbp3 = new MimeBodyPart();
                                mbp3.setText(msgText3);
//                              create the Multipart and its parts to it
                                Multipart mp = new MimeMultipart();
                                mp.addBodyPart(mbp0);
                                mp.addBodyPart(mbp1);
                                mp.addBodyPart(mbp2);
                                mp.addBodyPart(mbp3);
//                              add the Multipart to the message
                                msg.setContent(mp);
//                              send the message
                                Transport.send(msg);
                        }
                        catch (MessagingException mex){
                                mex.printStackTrace();
                                Exception ex = null;
                                if ((ex = mex.getNextException()) != null) {
                                        ex.printStackTrace();
                                }
                        }
                }
}
```

Servlet MostraEmail2 – exibe a mensagem original am resposta a uma requisição http.

```
package acfos;

// Classe testes leitura email com multiplos attachaments
// Observe que doRetrieve() pode ser chamado recursivamente para tratar os diversos
attachements (anexos)

// Codigo adapdado a partir de uma das classes servlet do artigo do Gamelan
(www.developer.com) :
// WebMail in Java: Reading E-mail - Benoît Marchal
// http://www.developer.com/java/other/article.php/618481
//
// Principais alterações:
// 1 - Implementado tratamento de InputStream no lugar do acesso direto POP3
// 2 - Aperfeiçoamento no tratamento dos anexos
// 3 - Alteração do Streamming dos Anexos para ser feito através de Buffered readres
// 4 - Aparentemente para mostrar o anexo (Attachment) com o encoding correto parece ser
atrapalhado
// 5 - Pela criação de outros Writers/OutputScreen


import java.io.*;
//import java.net.*;
import java.text.*;
```

```
import javax.crypto.SecretKey;
import javax.mail.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.util.Properties;
import javax.mail.Internet.*;


public class MostraEmail2 extends HttpServlet {

  private String path="";

  public void init(ServletConfig config) throws ServletException {
          super.init(config);
          path = config.getInitParameter("message_path");
  }

 // Globais para os Formulários.

  protected static final String listMsgHeader = "";
    //"<HTML><HEAD><TITLE>WebPOP3</TITLE></HEAD>" +
    //"<BODY><P><A HREF=\"{0}?action=list\">[ Anterior ]" +
    //"</A> <A HREF=\"{0}?action=logout\">[ Próximo ]" +
    //"</A>";


  protected static final String AttachmentHeader =
    "<HTML><HEAD><TITLE>Anexo</TITLE>";

   protected static final String AttachmentFooter =
    "</HEAD><BODY></BODY></HTML>";


  protected static final String listMsgFooter =
    "</BODY></HTML>";

  protected static final String linkList =
    "<P><FORM ACTION=\"{0}\" METHOD=\"POST\">" +
    "<A HREF=\"{0}?action=retrieve&email={1}\">" +
    "Message: {2}</A> " +
    "<INPUT TYPE=\"HIDDEN\" NAME=\"action\"" +
    " VALUE=\"delete\"><INPUT TYPE=\"HIDDEN\"" +
    " NAME=\"msg\" VALUE=\"{1}\">" +
    "<INPUT TYPE=\"SUBMIT\" VALUE=\"Delete\"></FORM>";

  protected static final String attachmentLink =
    "<A HREF=\"{0}?action=retrieve&email={1}\"TARGET=\"{0}\">{2}</A>";

 public void doGet(HttpServletRequest request, HttpServletResponse response)
```

```
  throws ServletException, java.io.IOException
  {

        //java.io.PrintWriter out = response.getWriter( );
        //out.print("<html>Teste</html>");
        //return;

    try {
      doRetrieve(request,response);

    } catch (Exception e) {
        java.io.PrintWriter out = response.getWriter( );
        response.setContentType("text/html");
        out.println("<html><head><title>Leitura Email HTTP</title></head><body>");
        out.println("<h2> Erro Acessando Mensagens Email HTTP." + e.getMessage() +
"</h2>");
        out.println("</br><h2> Caminho:" + path + "</h2>");
        out.println("</body></html>");

    }


  }//doGet



  // Classe auxiliares de imprimir o Formulário/Hyper link s
  protected void printForm(String form,
                    HttpServletRequest request,
                    HttpServletResponse response)
    throws IOException
  {
    PrintWriter writer = response.getWriter();
    form = MessageFormat.format(form,
      new Object[] { request.getServletPath() });
    writer.print(form);
    writer.flush();
  }

  protected void printHyperlink(String form,
                        HttpServletRequest request,
                        HttpServletResponse response,
                        String parameters,
                        String text)
    throws IOException
  {
                //request.getServletPath()
    PrintWriter writer = response.getWriter();
    form = MessageFormat.format(form,
      new Object[] { "mostra.acf",
```

```
                    parameters,
                    text });
    writer.print(form);
    writer.flush();
  }



  protected void doRetrieve(HttpServletRequest request,
                      HttpServletResponse response)
    throws ServletException, IOException,
          MessagingException
  {

    // A mensagem de Email está gravada em um arquivo cujo nome é passado como
parâmetro
    // Neste ponto foi alterada a classe de obter Email para obter de um arquivo

      String emailFile =  request.getParameter("email");

      //String eml  =  getServletContext( ).getRealPath("/") + "\\emails\\" + emailFile;

      // Esta fixado, mas vai ser alterado para uma outra logica.
      //String eml = "D:\\Tomcat-5.5.16\\webapps\\emails\\" + emailFile;
         //String eml = "C:\\Arquivos de programas\\jakarta-tomcat-5.5.7\\webapps\\emails\\" +
emailFile;
         String eml = path + emailFile;

       //Debug
       //java.io.PrintWriter out = response.getWriter( );
       //out.println("Abrindo o email arquivo=" + emailFile + " eml=" + eml + "<BR>");
       //out.flush();



      Properties props = System.getProperties();
      props.put("mail.host", "smtp.dummydomain.com");
      props.put("mail.transport.protocol", "smtp");

      Session mailSession = Session.getDefaultInstance(props, null);

                  SecretKey key = loadKey();
      // Create encrypter/decrypter class
      DesEncrypter encrypter = new DesEncrypter(key);

      // Decrypt
      encrypter.decrypt(new FileInputStream(path + emailFile + ".crp"),
         new FileOutputStream(path + emailFile));   //+".crp"
```

```
    File emlFile = new File(eml); //Abre a mensagem
    FileInputStream source = null;

    try {
      source = new FileInputStream(emlFile);
    }
    catch (Exception e) {

      e.printStackTrace();

      java.io.PrintWriter out = response.getWriter( );
      out.println("Erro localização arquivo -- file=" + emailFile + " eml=" + eml + "<BR>");
      out.flush();
      return;
    }

    MimeMessage msg = new MimeMessage(mailSession, source);


    if(msg == null)
      return;


  if(request.getParameter("part") == null)
    writeMessage(msg,request,response);
  else
    writePart(msg,request,response);
}

protected void writeEnveloppe(Message msg,
                      HttpServletRequest request,
                      HttpServletResponse response)
  throws IOException, MessagingException
{
  printForm(listMsgHeader,request,response);
  PrintWriter writer = response.getWriter();
  writer.println("<P><B>De:</B> ");
  writeAddresses(msg.getFrom(),writer);
  writer.println("<B>Para:</B> ");
  writeAddresses(
    msg.getRecipients(Message.RecipientType.TO),
    writer);
  writer.println("<B>Cc:</B> ");
  writeAddresses(
    msg.getRecipients(Message.RecipientType.CC),
    writer);
  writer.println("<B>Assunto:</B> ");
  writer.println(msg.getSubject() + "</P>");
  writer.flush();
}
```

```
protected void writeAddresses(Address[] addresses,
                              PrintWriter writer)
{
  if(addresses != null)
    for(int i = 0;i < addresses.length;i++)
      writer.println(addresses[i] + "<BR>");
  else
    writer.println("<BR>");
}

protected void writeMessage(Message msg,
                            HttpServletRequest request,
                            HttpServletResponse response)
  throws ServletException, IOException,
       MessagingException
{


  String emailFile =  request.getParameter("email");

  PrintWriter writer = response.getWriter();
  writeEnveloppe(msg,request,response);
  if(msg.isMimeType("multipart/*"))
  {
    Multipart multipart = (Multipart)msg.getContent();
    for(int i = 0;i < multipart.getCount(); i++)
    {
      Part p = multipart.getBodyPart(i);
      if(p.isMimeType("text/plain"))
      {
        writer.print("<PRE>");
        writer.print(p.getContent());
        writer.print("</PRE>");
      }
      else
      {
        String filename = p.getFileName();
        printHyper link(attachment Link,
                        request,response,
                        emailFile +
                        "&part=" + i,
                        "Anexo " + i + ":" + filename + "<BR>");
      }
    }
  }
  else if(msg.isMimeType("text/plain"))
  {
    writer.print("<PRE>");
    writer.print(msg.getContent());
```

```
      writer.print("</PRE>");
    }
    else
      printHyperlink(attachmentLink,
                     request,response,
                     emailFile +
                     "&part=-1",
                     "Conteúdo Anexo");
    writer.flush();
  }

  protected void writePart(Message msg,
                           HttpServletRequest request,
                           HttpServletResponse response)
    throws IOException, MessagingException, ServletException
  {
    int partnr = Integer.parseInt(request.getParameter("part"));

    Part p;
    if(partnr < 0)
      p = msg;
    else
    {
      Multipart multipart = (Multipart)msg.getContent();
      p = multipart.getBodyPart(partnr);
    }


    response.setContentType(p.getContentType());

    if(p.getFileName() != null)
      response.setHeader("Content-Disposition","attachment; filename=\"" + p.getFileName()
+ "\"");

    OutputStream out = response.getOutputStream();
    InputStream in = p.getInputStream();

    /*
    int c = in.read();
    while(c != -1)
          {
      out.write(c);
      c = in.read();
    }
    */

    //PrintWriter writer = response.getWriter();


    BufferedInputStream buf = new BufferedInputStream(in);
```

```
    int readBytes = 0;

    try {



      //read from the file; write to the ServletOutputStream
      while((readBytes = buf.read( )) != -1)
      out.write(readBytes);


    } catch (Exception e){

        throw new ServletException("Erro processando Anexo:" + e.getMessage( ));
    } finally {

    //close the input/output streams
    if(out != null)
       out.close( );
    if(buf != null)
       buf.close( );
    }
  }

        public SecretKey loadKey() {
                SecretKey chave = null;
                try {
                        FileInputStream in = new FileInputStream(path + "base.key");
                        ObjectInputStream s = new ObjectInputStream(in);
                        chave = (SecretKey) s.readObject();
                } catch (ClassNotFoundException e) {
                        System.err.println("Erro ao salvar a chave " + e.getMessage());
                } catch (FileNotFoundException e) {
                        System.err.println("Erro ao salvar a chave " + e.getMessage());
                } catch (IOException e) {
                        System.err.println("Erro ao salvar a chave " + e.getMessage());
                }
                return chave;
        }


} // Fim ReadMail
```

Classe DesEncrypter – responsável pela criptografia.

```
package acfos;

import java.io.InputStream;
import java.io.OutputStream;
```

```
import javax.crypto.Cipher;
import javax.crypto.CipherInputStream;
import javax.crypto.CipherOutputStream;
import javax.crypto.SecretKey;
import javax.crypto.spec.IvParameterSpec;

public class DesEncrypter {
   Cipher ecipher;
   Cipher dcipher;
   private IvParameterSpec paramSpec;

   DesEncrypter(SecretKey key) {
      // Create an 8-byte initialization vector
      byte[] iv = new byte[]{
         (byte)0x8E, 0x12, 0x39, (byte)0x9C,
         0x07, 0x72, 0x6F, 0x5A
      };
      paramSpec = new IvParameterSpec(iv);
      try {
         ecipher = Cipher.getInstance("DES/CBC/PKCS5Padding");
         dcipher = Cipher.getInstance("DES/CBC/PKCS5Padding");

         // CBC requires an initialization vector
         ecipher.init(Cipher.ENCRYPT_MODE, key, paramSpec);
         dcipher.init(Cipher.DECRYPT_MODE, key, paramSpec);
      } catch (java.security.InvalidAlgorithmParameterException e) {
      } catch (javax.crypto.NoSuchPaddingException e) {
      } catch (java.security.NoSuchAlgorithmException e) {
      } catch (java.security.InvalidKeyException e) {
      }
   }

   // Buffer used to transport the bytes from one stream to another
   byte[] buf = new byte[1024];

   public void encrypt(InputStream in, OutputStream out) {
      try {
         // Bytes written to out will be encrypted
         out = new CipherOutputStream(out, ecipher);

         // Read in the cleartext bytes and write to out to encrypt
         int numRead = 0;
         while ((numRead = in.read(buf)) >= 0) {
            out.write(buf, 0, numRead);
         }
         out.close();
      } catch (java.io.IOException e) {
      }
   }
```

```
   public void decrypt(InputStream in, OutputStream out) {
      try {
         // Bytes read from in will be decrypted
         in = new CipherInputStream(in, dcipher);

         // Read in the decrypted bytes and write the cleartext to out
         int numRead = 0;
         while ((numRead = in.read(buf)) >= 0) {
            out.write(buf, 0, numRead);
         }
         out.close();
      } catch (java.io.IOException e) {
         System.out.println("Erro: " + e.getMessage());
      }
   }
}
```

Classe Param – lê os parâmetros do arquivo de parâmetros

```
package acfos;

import java.io.IOException;
import java.util.Properties;

public class Param {
         /*
         *         Recupera os parâmetros da aplicação
         */
         public Properties getProperties() {

            Properties properties = new Properties();
            try {
               properties.load(this.getClass().getResourceAsStream("filter.properties"));
            } catch (IOException e) {
                  System.out.println("Erro ao carregar os parametros: " + e.getMessage());
            }
            return properties;
         }
}
```

Arquivo de parâmetros

```
# Parâmetros do FilterServer
in_por=3000
#msg_path=C:/filterserver/messages/
msg_path=C:/Arquivos de programas/jakarta-tomcat-5.5.7/messages/
```

```
#smtp=icarai.midiacom.uff.br
#web=http://www.midiacom.uff.br/~antonio/mostra.acf

smtp=smtp.dyn.com.br
web=http://localhost:8080/novomail/mostra.acf
```