INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

Mailspike

Henrique Caldeira Carvalho Aparício

Dissertação para obtenção do Grau de Mestre em

Engenharia Informática e Computadores

Júri

Presidente: Professor Doutor Alberto Manuel Ramos da Cunha
Orientador: Professor Doutor Carlos Nuno da Cruz Ribeiro
Co-orientador: Engenheiro João Gouveia
Vogal: Professor Doutor Ricardo Jorge Fernandes Chaves

Outubro de 2010

Agradecimentos

Agradeço ao Professor Carlos Ribeiro, cuja disponibilidade para ajudar foi impecável, prestando-me apoio e orientação sempre que necessário, não só nesta tese, mas noutras situações do meu percurso académico.

Agradeço ao Francisco Fonseca, que acreditou em mim para desenvolver este trabalho quando ainda não me conhecia. Agradeço também aos meus outros colegas da *AnubisNetworks*, nomeadamente ao Paulo Pacheco e ao Rodrigo Fernandes, que me ajudaram a vencer a inércia inicial e ao João Gouveia, que me orientou ao longo de todo o trabalho.

Finalmente agradeço aos meus pais, irmão, familiares e amigos, que me apoiaram sempre moralmente e acreditaram sempre em mim.

Resumo

Com o aumento de *spam* que circula na internet, é cada vez mais difícil ter filtros de *email* que consigam filtrar todo o *spam* atempadamente. Os filtros de conteúdos tradicionais, ainda que bastante eficazes a detectar *spam*, já não conseguem dar conta da quantidade enorme de *emails* que passam por eles. É portanto necessário encontrar soluções que aliviem a carga excessiva aos filtros tradicionais.

O *MailSpike* é um sistema da empresa *AnubisNerworks*, que utiliza técnicas de filtragem de *email* baseadas em identificadores, nomeadamente a técnica de *fingerprinting* e a técnica de reputação IP. Estas técnicas ajudam a tirar aos filtros tradicionais uma grande percentagem da carga de *emails* que têm de ser processados, pois conseguem classificar a maioria deles com um desempenho muito superior.

Neste trabalho apresentam-se algumas propostas para melhoria dos sistemas do *MailSpike* que utilizam as técnicas de *fingerprinting* e reputação, de forma a que estas consigam ter a capacidade necessária para processar a crescente quantidade de *emails* que chegam às organizações. Ambos os sistemas foram portanto re-implementados de raíz, de forma a obter mais desempenho e flexibilidade, e algumas melhorias nos algoritmos utilizados por eles.

Os resultados obtidos dos testes realizados com os novos sistemas, permitem verificar as melhorias obtidas em termos de performance, assim como as vantagens trazidas pela nova heurística de reputação e pelo novo algoritmo de detecção de *botnets*.

Palavras-chave

Email, *Spam*, *Fingerprinting*, Reputação, *Botnets*, Arquitecturas

Abstract

With the growing flow of spam among the internet, it is increasingly harder to have email filters that can take action on all spam, in useful time. Traditional content filters, though very effective in spam detection, can no longer respond to the massive amount of emails that go through them. It is therefore necessary to find solutions that relieve the excessive load from traditional filters.

MailSpike is a system developed by the AnubisNetworks organization, which uses email filtering techniques based on identifiers, namely the fingerprinting technique and the sender reputation technique. These techniques can take a great percentage of the email load from traditional filters, because they can classify most of them with significantly superior performance.

In this thesis we present some proposals to improve the MailSpike fingerprinting and reputation systems, so that they can have the necessary capacity to process the growing amount of emails received by organizations. Both systems were therefore re-designed and re-implemented, in order to obtain better performance and flexibility, along with some improvements in their algorithms.

The obtained results from the tests that were made to the new systems, allow us to verify the obtained improvements in terms of performance, as well as the advantages brought by the new reputation heuristic and the new botnet detection algorithm.

Keywords

Email, Spam, Fingerprinting, Reputation, Botnets, Architectures

Conteúdo

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

Acrónimos

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ASN	Autonomous System Number
BGP	Border Gateway Protocol
DKIM	DomainKey Identified Email
DNS	Domain Name System
DNSBL	Domain Name System Blackhole List
DSN	Delivery Status Notification
HTML	HyperText Markup Language
IP	Internet Protocol
ISP	Internet Service Provider
MD5	Message Digest 5
MIME	Multipurpose Internet Mail Extention
MTA	Mail Transfer Agent
MX	Mail eXchange record
NDR	Non Delivery Report
OCR	Optical Character Recognition
P2P	Peer to Peer
PTR	PoinTeR record
RFC	Request For Comments

SHA Secure Hash Algorithm

SMTP Simple Mail Transfer Protocol

SPF Sender Policy Framework

SQL Structured Query Language

UDP User Datagram Protocol

URI Uniform Resource Identifier

URL Uniform Resource Locator

Capítulo 1

Introdução

1.1 Motivação

O *email* é desde há vários anos, uma ferramenta de cada vez maior importância enquanto meio de comunicação. A sua crescente utilização levou inevitavelmente à sua exploração por parte de entidades que o utilizam para enviar publicidade, vírus e *phishing*, para o máximo número possível de destinatários. A principal teoria que explica o envio de *spam*, é que numa enorme quantidade de pessoas que o recebem, irá sempre haver uma pequena minoria que estará interessada no conteúdo daquelas mensagens, ou que irá simplesmente clicar nalgum *link* que esteja incluído na mensagem. O envio de *spam* já atingiu proporções tais, que cerca de 90% dos *emails* que chegam às empresas e a utilizadores individuais, são *spam*. Isto é portanto um problema que pode afectar gravemente a produtividade das empresas, e para o qual é necessário encontrar soluções cada vez mais eficientes para o combater.

Uma das principais técnicas para combater o *spam*, é a utilização de diferentes filtros, que analisam o conteúdo dos *emails* e que de acordo com o resultado dessa análise, os classificam ou não como *spam*. No entanto, os algoritmos que efectuam esta análise são na maioria das vezes bastante pesados, o que pode ser um problema grave, uma vez que em empresas de grandes dimensões, a quantidade de *emails* recebidos é tal, que a capacidade de processamento não chega para processar os *emails* todos atempadamente. Isto faz com que seja necessária a utilização de técnicas que evitem todo este processamento de elevada carga.

Esta dissertação realizada na empresa *AnubisNetworks*, uma empresa especializada em segurança de *email*, pretende melhorar consideravelmente alguns sistemas previamente desenvolvidos pelos colaboradores mais antigos da empresa. Estes sistemas (no âmbito deste projecto, *Fingerprinting* e Reputação IP), têm como objectivo evitar que a maioria das mensagens recebidas, não tenham que ser sujeitas à pesada análise

dos filtros de conteúdos tradicionais, e permitem portanto rejeitar as mensagens assim que chegam ao servidor de *email* de destino.

A análise de texto, de imagem e de ficheiros de um *email* são técnicas utilizadas na detecção de *spam* que, apesar de bastante eficazes, são muito dispendiosas em termos de processamento. Além disso, este tipo de filtros de *email* necessitam de ser actualizados frequentemente, de forma a poderem adaptar-se às novas técnicas que os *spammers* utilizam.[R] Porém, existem técnicas que se baseiam em dados diferentes do conteúdo do *email* e que permitem evitar a utilização destes filtros, ficando estes apenas como último recurso. O projecto a ser desenvolvido nesta dissertação explora duas destas técnicas, que já são utilizadas na *AnubisNetworks*, mas que requerem um redesenho, de forma a melhor satisfazer os seus objectivos.

O *MailSpike* (ou apenas *Spike*) é um sistema da *AnubisNerworks*, que utiliza entre outras, a técnica de *Fingerprinting* de mensagens e a técnica de Reputação (*Sender Reputation*) para fazer filtragem de *email*. A primeira consiste de um modo geral em gerar uma assinatura (*hash*) a partir do corpo das mensagens, que por sua vez é guardada numa base de dados caso esta mensagem seja considerada *spam*. Desta forma, sempre que a mesma mensagem chega ao servidor de *mail*, esta é automaticamente rejeitada sem ter que ser sujeita aos filtros baseados no conteúdo das mensagens. A segunda técnica usa uma heurística que se baseia essencialmente no histórico de envio de mensagens dos emissores, para calcular um valor de reputação que é atribuído ao endereço IP, fazendo também detecção de *botnets* no processo. O valor de reputação de um IP e o facto de este pertencer ou não a uma *botnet* são factores que permitem decidir se um *email* enviado por esse IP deve ser aceite ou rejeitado.

A implementação actual destes dois sistemas (*fingerprinting* e reputação), ainda que bastante eficazes apresentam alguns problemas. Ambos têm problemas graves em termos de organização. A implementação em PERL existente, não facilita a modificação, extensão ou mesmo correcção de erros. Por exemplo, neste momento existe um projecto na *Anubis* que pretende adicionar novos tipos de assinaturas ao *Spike*, e para que estas possam ser suportadas, iria implicar alterações que seriam bastante complexas no sistema actual. Outro problema comum aos dois sistemas é o facto da capacidade de desempenho ser cada vez mais insuficiente para o volume de dados que tem de ser processado.

Existem ainda alguns problemas exclusivos a cada um dos sistemas. O sistema de *fingerprinting*, nalguns casos pouco frequentes tem dificuldades em capturar a mensagem original, acontecendo nesses casos que o texto usado para gerar a assinatura inclui partes que não era suposto serem incluídas. A heurística do sistema de reputação mencionada acima pode também ser consideravelmente melhorada. Neste momento é feito um simples incrementar de contadores de *ham* e *spam* por IP. Isto faz com que eventualmente um valor máximo seja atingido, obrigando a limpezas per-

iódicas da base de dados, e com que não haja forma de distinguir ou dar mais peso ao histórico mais recente, ou seja, um IP que tenha enviado um número muito elevado de *spam* teria que enviar pelo menos o mesmo número de mensagens *ham* para se poder redimir.

1.2 Objectivos

O sistema de *fingerprinting* actual do *Spike* tem alguns problemas na organização (pouca modularidade) e mesmo no funcionamento do algoritmo de captura da mensagem original.

Assim, o primeiro objectivo deste trabalho passa por fazer uma refactorização completa do sistema de *fingerprinting* previamente desenvolvido na *AnubisNetworks*, organizando a arquitectura da solução de maneira diferente. A nova solução deverá corrigir os problemas actuais do *Spike*, sem deixar de utilizar o protocolo actual, e acima de tudo deverá ter maior capacidade de desempenho, uma vez que todas as mensagens serão sujeitas a este filtro.

O sistema de reputação actual tem problemas na organização semelhantes aos do sistema de *fingerprinting* (implementação em PERL, pouco modular, capacidade de desempenho insuficiente) e a heurística tem alguns aspectos que podem ser consideravelmente melhorados. O segundo objectivo é portanto redesenhar a arquitectura do sistema de reputação, criando para isso um novo modelo de armazenamento de dados e um novo modelo de cálculo da reputação. Com isto, espera-se conseguir mais facilidade de alteração e manutenção do sistema, maior capacidade de processamento dos dados e cálculo de valores de reputação mais justos.

1.3 Contribuições

O foco desta tese é o redesenho e re-implementação de raiz de dois componentes do sistema *MailSpike* da *AnubisNetworks*.

Inicialmente é feita uma análise do sistema actual, de forma a perceber onde se podem introduzir melhorias, seguida da projecção e implementação do novo sistema, que deve ser mais performante e flexível em relação ao anterior.

As contribuições desta tese podem ser enumeradas da seguinte forma:

- Introdução de duas novas camadas de abstracção no sistema de *fingerprinting*, que o vêm tornar mais flexível e eficiente;
- Novo algoritmo de preprocessamento dos *emails* para posterior geração de assinatura;
- Novo modelo de base de dados de reputação, mais simples e flexível;

- Redesenho total da arquitectura do sistema de reputação, com mais flexibilidade e mais performance;
- Pequena alteração na heurística de reputação de forma a utilizar médias móveis exponenciais para efectuar os cálculos de reputação;
- Novo algoritmo de detecção de *clusters* de *zombies*;

1.4 Organização do Documento

Este documento encontra-se dividido em 6 secções onde é contextualizado e explicado o que foi realizado no trabalho desenvolvido.

A presente secção deste relatório começa por fazer uma introdução, onde são descritos a motivação e os objectivos e contribuições deste trabalho.

A segunda secção apresenta algum trabalho relacionado com o que foi realizado neste projecto. Nesta secção são também explicados os vários tipos de filtros de *spam* existentes e alguns conceitos relevantes para a detecção de *spam*.

Na terceira secção é apresentada a arquitectura do *Spike*, que é o sistema onde se insere o trabalho desenvolvido nesta tese. Aqui é feita uma descrição geral da organização do *Spike*, e dos vários componentes que o constituem.

A quarta secção começa por descrever o modo como o trabalho foi desenvolvido. Nas subsecções seguintes apresenta-se pormenorizadamente o trabalho desenvolvido nesta dissertação.

Na quinta secção são discutidos os resultados do trabalho desenvolvido.

Para finalizar, na sexta secção apresentam-se as conclusões tiradas do desenvolvimento desta dissertação, e algum possível trabalho futuro.

Os algorítmos envolvidos neste trabalho são apresentados nos anexos do documento.

Capítulo 2

Estado da Arte

Neste capítulo começa-se por introduzir os vários tipos de filtros de forma a contextualizar o problema que este trabalho procura resolver, seguido de alguns conceitos relevantes para a detecção de *spam* na secção 2.2. De seguida, nas secções 2.3 e 2.4 são explicadas em mais detalhe as técnicas de *fingerprinting* e reputação do emissor, e são apresentados alguns sistemas que utilizam estas técnicas. Finaliza-se o capítulo com uma discussão onde se comparam os vários sistemas mencionados nas subsecções 2.3 e 2.4.

2.1 Tipos de filtros

A figura 2.1[10] procura dar uma visão geral dos vários tipos de filtros de *email* existentes de acordo com a informação utilizada para fazer a filtragem.

Os filtros de *email* podem dividir-se em três tipos principais: filtros *Whitelists/Blacklists*, filtros de análise de conteúdo e filtros baseados em identificadores. Os primeiros consistem de simples listas de emissores cujas mensagens são automaticamente aceites (se pertencerem à *whitelist*) ou automaticamente rejeitadas (caso pertençam à *blacklist*). Os filtros de análise de conteúdo, podem-se dividir em filtros de análise de texto, análise de imagem ou filtros estatísticos. Estes filtros fazem uma análise exaustiva do texto contido na mensagem[1] de forma a encontrar palavras que possam indicar que a mensagem é *spam*. Os últimos filtros, filtros baseados em identificadores, podem-se dividir de acordo com o tipo de identificador utilizado: identificadores de endereço e identificadores de conteúdo. Os identificadores de endereço podem ser endereços de *email* dos emissores, o domínio dos emissores ou endereços IP dos emissores e são tipicamente usados em filtros que calculam a reputação dos emissores. Os identificadores

[1] Tipicamente os filtros de análise de imagem fazem uma análise OCR (*Optical Character Recognition*) e de seguida são aplicados os algoritmos de análise de texto normais.

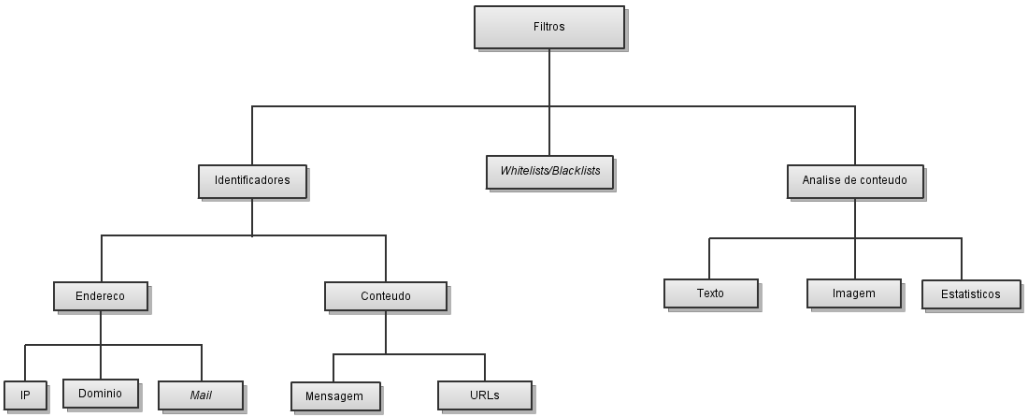

Figura 2.1: Tipos de filtros

de conteúdo podem ser a própria mensagem (assinatura da mensagem) ou URLs que apareçam na mensagem e são tipicamente usados em filtros de *fingerprinting* de mensagens.

A imagem refere três tipos de filtros principais, embora existam outros que são menos usados e menos relevantes no contexto deste trabalho.

2.1.1 *Whitelists/Blacklists*

Este é um tipo de filtro bastante simples que se limita a listar endereços de *email* ou IP, que se sabe serem típicos emissores de *ham* ou *spam*. No caso das *Whitelists*, todos os *emails* são rejeitados com excepção dos *emails* provenientes de fontes que constam da lista. Ao contrário destas, as *Blacklists* aceitam todos os *emails*, excepto os de provenientes de fontes listadas na *blacklist*.

Apesar da sua simplicidade, este tipo de filtro é bastante usado. Isoladamente estes filtros são demasiado impeditivos (no caso das *whitelists*) ou demasiado permissivos (no caso das *blacklists*), mas quando usados em conjunto com outros tipos de filtros, podem ajudar a reduzir o número de falsos negativos e falsos positivos.[7]

2.1.2 Filtros de análise de conteúdo

Os filtros baseados no conteúdo da mensagem, são ainda hoje dos mais usados na classificação de *email*. Nesta categoria de filtros, podemos incluir filtros de análise de texto e filtros estatísticos.

Os filtros de análise de texto procuram na mensagem, palavras que são tipicamente usadas por *spammers* e classificam o *email* como *spam* caso estas sejam encontradas.[7] As técnicas mencionadas mais abaixo conseguem iludir estes filtros com facilidade, sendo portanto necessário um esforço de processamento adicional para as conseguir combater.

Os filtros estatísticos, ou mais particularmente, os filtros Bayesianos, utilizam a teoria desenvolvida pelo matemático Thomas Bayes que permite calcular a probabilidade de ocorrência de um evento, baseada nas probabilidades de dois ou mais acontecimentos independentes. Assim, estes filtros verificam a frequência de ocorrência das palavras em mensagens *ham* e mensagens *spam*, e vão aprendendo à medida que processam mais mensagens a classificar cada vez melhor os novos *emails*.[7, 11]

Este tipo de filtros, apesar de serem bem sucedidos a classificar a maioria das mensagens, sofrem de alguns problemas que podem prejudicar fortemente a identificação do spam. Um dos problemas é o facto de muitas vezes não haver capacidade de processamento suficientemente alta para classificar todas as mensagens em tempo útil, especialmente em ambientes empresariais onde o volume de mensagens é extremamente elevado. Outro problema é a facilidade com que os *spammers* se adaptam a estes filtros, fazendo uso de várias técnicas que facilmente conseguem induzi-los em erro. De seguida, descrevem-se resumidamente algumas destas técnicas usadas pelos *spammers*.[9]

2.1.2.1 Técnicas de Spam

Dictionary attack ou *Word salad*, é o termo utilizado para a técnica em que os *spammers* acrescentam à mensagem palavras que não fazem sentido no contexto em que são inseridas. Estas palavras que são adicionadas ao conteúdo da mensagem, são quase sempre palavras consideradas boas, no sentido em que são palavras que normalmente aparecem nas mensagens *ham*. A ideia por trás desta técnica, é tentar subir a pontuação que os classificadores atribuem à mensagem.[9, 12]

Frequent word attack ou *Light reading*, funciona como a técnica anterior, mas um pouco melhorada. Em vez das "palavras boas" serem adicionadas aleatoriamente no corpo da mensagem, são adicionadas frases completas e com significado além da mensagem que o *spammer* quer passar. Isto faz com que seja ainda mais difícil para os classificadores excluir as palavras que estão a mais, levando a uma pontuação mais favorável para o *spammer*. Esta técnica e a anterior são utilizadas principalmente para enganar os filtros estatísticos.[9, 12]

Tiny Text é uma técnica que consiste em fazer com que o conteúdo da mensagem seja legível para a pessoa que a lê, mas não pelo filtro de *spam*. Isto é con-

seguido introduzindo caracteres entre as palavras, com tamanho de fonte ou cor diferentes do resto do texto, o que dificulta a identificação das palavras que interessam por parte do filtro de *spam*.[9]

Scrabble Spam aproveita o facto de muitos filtros de *spam* dependerem das palavras estarem correctamente soletradas. É possível escrever as palavras incorrectamente soletradas, sem que estas deixem de ser legíveis pela pessoa que recebe a mensagem - e.g. "Crteae a mroe ppsorerous fuutre for yuolserf".[9]

Ilusões ópticas é mais uma vez uma técnica em que a pessoa que recebe a mensagem consegue ler o texto sem dificuldade, enquanto que o filtro de *spam* tem problemas, uma vez que para escrever as "palavras más" são usados caracteres especiais que se assemelham a determinadas letras.[9]

2.1.3 Filtros baseados em Identificadores

Para compreender este tipo de filtros, é necessário primeiro perceber a noção de identificador neste contexto. Considera-se um identificador, um aspecto inerente à mensagem enviada, que de alguma forma está fortemente correlacionada com a mensagem.[10] Estes identificadores podem ser usados não só para classificar as mensagens, mas também para saber qual classificação previamente atribuída às mensagens. Os identificadores que são utilizados neste tipo de filtros podem ser essencialmente de dois tipos: de endereço e de conteúdo.

Os identificadores de endereço, podem ser por exemplo o endereço IP do emissor, o endereço de *mail* do emissor ou o domínio do emissor. Na classificação de *emails* uma técnica que normalmente faz mais uso deste tipo de identificadores, é o cálculo da reputação do emissor.

Os identificadores de conteúdo consistem em aspectos que normalmente constam do corpo da mensagem, como por exemplo URLs ou mesmo uma *hash* (assinatura) da própria mensagem. Neste caso, a técnica mais frequentemente usada, é a técnica de *fingerprinting*.

É no âmbito dos filtros baseados em identificadores que se insere esta dissertação. De seguida são apresentadas algumas técnicas usadas por *spammers* para iludir estes filtros e também algum trabalho relacionado acerca deste tipo de filtros.

2.1.3.1 Técnicas de *spam*

Botnets Conjuntos de máquinas (Zombies) comprometidas com software malicioso, que têm como objectivo enviar mensagens de spam para um grande número de destinatários.[9] Os zombies podem fazer parte de clusters que estão sob o controlo de uma única entidade, ou podem ser máquinas de end-users que após

terem sido infectadas, tratam elas próprias de enviar as mensagens de spam. As mensagens enviadas pelos zombies, apesar de bastante semelhantes, podem conter pequenas diferenças que servem principalmente para iludir os sistemas de *fingerprinting*.[9]

Adaptação O cálculo da reputação dos emissores é um método de filtragem de *spam* cada vez mais usado. Uma maneira de os *spammers* dificultarem este processo é criar contas de *email* em vários ISPs ou mesmo ganhar acesso a contas de *email* comprometidas pertencentes a alguma organização. Com isto os *spammers* esperam obter valores de reputação previamente associados a emissores de *email* legítimo. Um *cluster* de *zombies* que esteja a infectar uma organização ou um computador pessoal, também irá ter o mesmo efeito.[9]

Autenticação mal intencionada A autenticação de *email* é um processo com pouca segurança, em que apenas se verifica se o domínio do endereço de *email* está de acordo com o endereço IP do servidor de *email* que é usado para enviar a mensagem. Com isto o receptor da mensagem consulta um *SPF Record* onde se encontram publicados os endereços IP permitidos para um certo domínio. Muitas organizações deixam esta opção em aberto, permitindo aos *spammers* a utilização do domínio de uma organização com qualquer endereço IP.[9]

2.2 Conceitos relevantes

Aqui são apresentados alguns conceitos relevantes na realização deste trabalho.

Spam Embora seja complicado definir uma barreira que faça a divisão entre os *emails* que são *spam* e os que não são, explica-se aqui a definição utilizada pela *Spamhaus,* que se considera ser a mais aceite. Um *email* que é não solicitado[2] pode não ser necessariamente considerado *spam*, assim como não há certeza que um *email* enviado em massa[3] para muitos recipientes seja *spam.* Para poder ser considerado *spam* com certeza, o *email* tem de obedecer às duas regras anteriores em simultâneo, ou seja, um *email* é considerado *spam*, se for não solicitado e enviado em massa (*Unsolicited Bulk Email).*[2]

Ham É o contrário de *spam*, ou seja, são *emails* legítimos que é suposto chegarem ao destinatário. Em contraste com a definição de *spam* anterior, pode-se definir *ham* como *email* que pode ser não solicitado ou enviado em massa, mas nunca os dois em simultâneo. Pode também não ter nenhuma destas duas características.

[2] Por exemplo, primeiros contactos, recrutamento de empregados
[3] Por exemplo, *mailing lists*, *newsletters*

Falso negativo Mensagem de *email* que apesar de na realidade ser *spam*, consegue passar por todos os filtros como se fosse uma mensagem legítima, chagando assim à caixa de entrada do destinatário quando não devia chegar.

Falso positivo Mensagem de *email* legítima que é incorrectamente classificada como *spam* e por consequência disso, não chega ao destinatário quando devia chegar. Normalmente o número de falsos positivos aumenta com a diminuição de falsos negativos e vice-versa, e sendo os falsos positivos mais problemáticos, é preferível ter uma maior taxa de falsos negativos.

NDR/DSN *Non-Delivery Report* ou *Delivery Status Notification*, também denominado *bounce message*, é um tipo de *email* que resulta do envio de outro *email*, mas que não chega ao destino (pode acontecer porque o endereço de destino não existe, por não ser possível chegar ao servidor de destino, etc). O emissor do *email* que não chega ao seu destino recebe assim um NDR, que contem uma mensagem do seu servidor de mail a dizer que o envio falhou juntamente com a mensagem originalmente enviada. É frequente aparecerem mensagens *spam* deste tipo, uma vez que por vezes os *spammers* alteram o cabeçalho *from* da mensagem com um endereço válido e o cabeçalho *to* com um endereço inválido.

Hash Uma *hash* ou assinatura, é o resultado de uma operação irreversível. Uma função de *hash* ao receber determinado *input*, gera uma *hash*, através da qual é impossível voltar ao *input* original. Isto é utilizado na técnica de *fingerprinting* para gerar assinaturas que identificam univocamente cada *email*, mas com a particularidade de gerar assinaturas iguais ou semelhantes, para mensagens muito parecidas. De notar que as funções de *hashing* geram assinaturas muito diferentes, por muito semelhantes que sejam os inputs, sendo portanto necessário criar algoritmos que permitam obter a propriedade anterior.

Reputação IP É o valor associado a cada endereço IP que envia *emails*. Existem várias abordagens para efectuar o cálculo deste valor, sendo na maioria dos casos utilizada uma função heurística que usa diversos inputs (que variam de implementação para implementação). Este valor é depois usado como indicador de confiança desse IP, definindo um limiar que diz a partir de que valor de reputação é que os *emails* de determinado IP devem ser aceites. Certas abordagens da técnica de *Sender reputation* utilizam outras características do emissor[4] diferentes do IP, às quais são associados os valores de reputação.

Zombie É um emissor comprometido ou mal intencionado, que normalmente faz parte de uma rede (*botnet*) em que os intervenientes enviam todos os mesmos

[4]Como por exemplo o domínio do emissor ou o endereço de *email[10]*

emails ou *emails* muito semelhantes, que se podem sempre considerar como *spam*.

2.3 *Fingerprinting*

A técnica de *fingerprinting* (impressão digital) começou originalmente por ser usada em seres humanos. Através de uma impressão digital deixada nalguma superfície, é possível saber com 100% de certeza quem tocou nessa superfície, uma vez que não existem duas pessoas com impressões digitais exactamente iguais. Isto pode obviamente ter várias utilidades, como identificar possíveis criminosos ou como forma de autenticação, mas no fundo a finalidade acaba sempre por ser a identificação de algo ou alguém.

Esta ideia pode ser aplicada em informática, através da utilização de uma função (função de *hashing*) que transforma um certo *input* (que pode ser por exemplo um ficheiro ou um texto), num conjunto de caracteres (*hash*, *digest* ou assinatura) de tamanho fixo e normalmente muito mais reduzido que o *input* original. Estas funções garantem que um determinado *input* irá sempre gerar a mesma assinatura, e através da assinatura é impossível chegar ao *input* original. Esta função também garante que dois *inputs* diferentes, por muito parecidos que sejam um com o outro, resultarão em duas assinaturas completamente diferentes[5]. MD5 e SHA são dois exemplos de algoritmos de *hashing* bastante populares.

As funções de *hashing* podem ter bastantes aplicações diferentes, inclusive como medida de segurança. No caso do sistema operativo Linux, por exemplo, é utilizado o algoritmo MD5 para gerar *hashes* das *passwords* dos utilizadores. Em vez de as *passwords* serem guardadas em claro, apenas são guardadas as *hashes* das *passwords*, fazendo com que seja muito difícil alguém descobrir as *passwords*, uma vez que as funções de *hashing* são irreversíveis. Outras utilizações das funções de *hashing* incluem verificação de integridade na transferência de ficheiros (*checksum*), geração e validação de assinaturas digitais, entre outras.

No contexto deste trabalho, a função de *hashing* é utilizada não como medida de segurança, mas sim para gerar assinaturas a partir da mensagem contida nos *emails* recebidos. Desta forma, é possível identificar de forma suficientemente única todos os *emails* recebidos. A maior parte dos sistemas de *fingerprinting* de mensagens mantém uma base de dados, construída de forma colaborativa, com todas as assinaturas resultantes de *emails* classificados como *spam*. Sempre que um cliente que utiliza um destes sistemas recebe um novo *email*, apenas tem de verificar se a assinatura que representa esse *email* consta da base de dados, e em caso afirmativo, o *email* é

[5]Existe a possibilidade de haver colisão, ou seja, dois *inputs* diferentes resultarem na mesma assinatura, embora a probabilidade de isto acontecer ser extremamente reduzida. Esta probabilidade difere entre os vários algoritmos de *hashing*.

descartado. Os URLs são outro aspecto que pode estar contido na mensagem e que pode ajudar a identificar os *emails*, bastando para isto que a base de dados também guarde URLs, que sejam conhecidos por aparecer em certos *emails* classificados como *spam*. *Emails* que contenham esses URLs, podem também vir a ser descartados.

Estes filtros tornam-se mais eficientes a filtrar *spam* com o aumento do número de utilizadores, pois esse aumento implica mais gente a contribuir para o catálogo de *spam* e portanto maior probabilidade de uma mensagem *spam* já estar na base de dados quando é feita uma consulta.

De seguida são apresentados alguns sistemas de *fingerprinting* de mensagens, as suas características e o modo como funcionam.

2.3.1 *Vipul's Razor*

O *Vipul's Razor* (ou apenas *Razor*) é uma rede distribuída e colaborativa de detecção e filtragem de *spam*, que utiliza a técnica de *fingerprinting* de mensagens.[8] Este sistema segue a mesma ideia base mencionada acima, ou seja, uma base de dados construída através da contribuição dos utilizadores do sistema, onde são guardadas entre outras coisas, as assinaturas das mensagens consideradas *spam*, de maneira a que um utilizador que receba *spam* previamente recebido por outros, possa descartar logo o *email*.

A característica mais diferenciadora deste sistema em relação aos outros sistemas do mesmo tipo, é o mecanismo de geração de assinaturas. O *Razor* utiliza um tipo de assinaturas especial a que o autor do sistema (Vipul) denominou de assinaturas efémeras. Para gerar estas assinaturas, ao invés de aplicar o algoritmo de *hashing* a todo o texto da mensagem, apenas uma porção do texto é usada como *input* para a geração da assinatura. Esta porção de texto é escolhida com base num número aleatório, que é constantemente alterado e gerado colaborativamente pelos utilizadores do *Razor*, fazendo com que uma assinatura correspondente a um *email* seja diferente em dois instantes de tempo distintos. Para que isto seja possível, além das assinaturas, o *Razor* também guarda o corpo das mensagens, para que ao ser gerado novo número aleatório, as assinaturas que já se encontram na base de dados possam ser novamente geradas a partir de uma porção de texto diferente. Isto serve essencialmente como medida de segurança, para que os *spammers* não tenham maneira de saber que partes do texto serão usadas para gerar as assinaturas após geração de novo número aleatório. Sem esta técnica, um *spammer* poderia por exemplo encontrar uma mensagem *spam* que gerasse a mesma assinatura que uma mensagem legítima, podendo assim fazer com que o *Razor* começasse a adicionar na base de dados assinaturas de mensagens legítimas.[8]

Para garantir que as assinaturas são geradas a partir do texto da mensagem, o *Razor* tem uma componente que são os preprocessadores. Muitos *emails* antes de

serem enviados, são codificados em Base64 ou *Quoted-Printable*, para que seja possível incluír na mensagem caracteres que não fazem parte da língua Inglesa, e que portanto fazem uso de mais 1 bit em relação ao ASCII. Além disto também é normal que parte ou a totalidade do corpo dos *emails* seja enviada em HTML, para que a mensagem apareça formatada da maneira pretendida pelo emissor. Os preprocessadores do *Razor* fazem a descodificação das mensagens em Base64 e *Quoted-Printable*, e extraem a mensagem das partes em HTML, garantindo assim que apenas o texto original é usado para fazer a geração das assinaturas.[8]

O *Razor* possui também vários motores de filtragem, cada um responsável por diferentes tipos de assinaturas. Neste momento o *Razor* inclui quatro motores:

- VR1 - Equivalente à primeira versão do *Razor* de forma a suportar retro-compatibilidade;
- VR2 - Baseado em assinaturas SHA1 simples (geradas simplesmente a partir do texto inteiro);
- VR3 - Baseado em assinaturas *Nilsimsa*, ou seja, assinaturas *fuzzy*, que têm a capacidade de ignorar pequenas alterações do texto;
- VR4 - Baseado nas assinaturas efémeras mencionadas acima;

Esta característica permite adicionar novos motores ao sistema sem grande dificuldade em eventuais novas versões.[8]

Uma vez que a submissão das assinaturas para a base de dados é manual, no sentido em que depende inteiramente dos utilizadores do serviço, o *Razor* inclui um sistema de reputação e confiança denominado TeS (Truth Evaluation System). Este sistema atribui valores de confiança (entre 0 e 100) aos utilizadores que reportam/revogam assinaturas e às próprias assinaturas. Com isto, os utilizadores podem configurar um limiar que considerem ser aceitável para os valores de confiança das assinaturas, e caso uma assinatura esteja abaixo desse limiar, a mensagem não é rejeitada. Este sistema serve principalmente para eliminar falsos positivos.[8]

De seguida descrevem-se as principais acções que os utilizadores do *Razor* podem efectuar[8]:

- `razor-admin` - Acção que permite efectuar tarefas de administração, nomeadamente criar o registo que irá permitir reportar e revogar assinaturas (sem o registo um utilizador não pode ter valor de confiança associado).
- `razor-check` - É a acção que a partir de um *email* gera uma assinatura e verifica se esta consta da base de dados do *Razor*.

- `razor-report` - Acção que serve para reportar assinaturas de *emails* que o utilizador considera serem *spam*. Para poder efectuar esta acção, o utilizador tem que estar registado na rede, o que é conseguido através da acção `razor-admin`.
- `razor-revoke` - Acção que serve para revogar assinaturas de mensagens que o utilizador considera não serem *spam*. Para poder efectuar esta acção, o utilizador tem que estar registado na rede, o que é conseguido através da acção `razor-admin`.
- `razor-whitelist` - Lista de endereços de *email* (emissores) e assinaturas que o utilizador considera serem legítimos, ou seja, todos os *emails* provenientes destes emissores ou que gerem estas assinaturas, serão automaticamente aceites.

2.3.2 *Pyzor*

Até há relativamente pouco tempo, o *Pyzor* era apenas uma implementação do *Razor*, com a principal diferença de estar escrito na linguagem *Python* (em contraste com o *Razor* que está escrito em *Perl*). O código fonte do cliente do *Razor* é completamente aberto, mas o código fonte do servidor não é, e isso levou a que os autores do *Pyzor* decidissem implementar um novo protocolo. O processo de geração de assinaturas do *Pyzor* é o seguinte[6]:

1. Descartar cabeçalhos das mensagens
2. Se a mensagem tiver mais de quatro linhas:
   (a) Descartar os primeiros 20% da mensagem
   (b) Aproveitar as próximas 3 linhas
   (c) Descartar os próximos 40% da mensagem
   (d) Aproveitar as próximas 3 linhas
   (e) Descartar o resto da mensagem
3. Remover palavras (ou sequências de caracteres) com 10 ou mais caracteres
4. Remover endereços de *email*
5. Remover URLs
6. Remover tags HTML
7. Remover espaços brancos
8. Descartar linhas com menos de 8 caracteres

Esta forma de gerar assinaturas é ligeiramente parecida com o esquema de assinaturas efémeras do *Razor*, no sentido em que nem todo o texto da mensagem é usado para gerar as assinaturas. No entanto, o esquema de geração de assinaturas do *Pyzor* não muda com o tempo, como é o caso do esquema do *Razor*. Este esquema embora seja previsível para os *spammers*, mantém uma possibilidade de para certas mensagens com ligeiras modificações, serem geradas assinaturas iguais, embora isto não seja garantido.

Neste momento o *Pyzor* apresenta as mesmas funcionalidades que o *Razor*, com excepção da rede de confiança (TeS)[6], e da consulta de uma *Whitelist* antes de verificar a classificação da mensagem.[6]

De seguida são apresentadas algumas das acções mais relevantes que o *Pyzor* permite realizar[6]:

- `check` - Análogo ao `razor-check.`
- `report` - Análogo ao `razor-report`.
- `whitelist` - Permite enviar para o servidor assinaturas de mensagens para serem adicionadas à *Whitelist*.
- `discover` - Encontra servidores do *Pyzor*.
- `ping` - Verifica se um servidor está operacional.
- `digest` - Dado um *email* válido, escreve como *output* a assinatura correspondente a esse *email*.
- `predigest` - Dado um *email* válido, escreve como *output* as linhas da mensagem que são usadas para gerar uma assinatura.

2.3.3 DCC (*Distributed Checksum Clearinghouse*)

O DCC é mais um sistema como os mencionados anteriormente, em que os utilizadores podem verificar se uma mensagem *spam* já foi recebida por outros, podendo portanto descarta-la automaticamente se for esse o caso. Existe no entanto uma diferença bastante significativa em relação ao *Razor* e ao *Pyzor*.

No caso do *Razor* e do *Pyzor*, é frequente os clientes de *mail* dos utilizadores estarem configurados para verificar automaticamente para cada *email* recebido se a assinatura correspondente consta do catálogo de assinaturas reportadas por outros utilizadores. Quando um *email* que o utilizador considera ser *spam* não é apanhado desta forma, cabe ao utilizador reportar a assinatura dessa mensagem para o catálogo

[6]O facto de não haver rede de confiança, invalida a necessidade dos utilizadores terem de se registar para reportar assinaturas.

de assinaturas. Este processo, em conjunto com o *Truth evaluation System* (no caso do *Razor*) garante com bastante fiabilidade que as assinaturas contidas na base de dados, são apenas de mensagens *spam* (ou seja, enviadas em massa e não solicitadas).

Tal como nos sistemas anteriores os clientes de *mail* dos utilizadores do DCC estão configurados para verificar se as assinaturas resultantes de todos os *emails* recebidos constam da base de dados. A grande diferença é que o DCC guarda todas as assinaturas que os utilizadores verificam, e contabiliza o número de vezes que cada assinatura foi verificada.[4] De notar que não existe a noção de reportar assinaturas de *emails* que os utilizadores considerem ser *spam*. Em vez disto, sempre que um utilizador verifica uma assinatura, é-lhe devolvido o número total de vezes que ele e outros utilizadores verificaram a mesma assinatura. Com isto, o cliente de *email* decide se a mensagem deve ser descartada caso este valor esteja acima de um limiar (*threshold*) definido pelo utilizador. Isto significa que o DCC apenas olha ao facto das mensagens terem ou não sido enviadas em massa, não havendo portanto distinção entre *email* solicitado e não solicitado.[4] Para que um utilizador do DCC possa receber *emails* de *newsletters*, *mailing lists* e outros serviços de *bulk mail* (mensagens envidas em massa), cada utilizador deve manter uma *whitelist* de emissores de *bulk mail* cujas mensagens são solicitadas.[4]

O processo de geração de assinaturas é um pouco semelhante ao do *Pyzor*, ou seja, o *input* da função de *hashing* não é o texto da mensagem inteira, mas sim apenas algumas partes da mensagem. Isto serve principalmente para que as assinaturas sejam *"fuzzy"*, ou seja, que haja uma possibilidade de mensagens parecidas gerarem a mesma assinatura. Para que duas mensagens semelhantes gerem a mesma assinatura, basta que as partes diferentes nas duas mensagens sejam as partes ignoradas. No caso do DCC, as assinaturas *fuzzy* estão desenhadas de forma a ignorar apenas diferenças que não tenham influência no significado da mensagem. O algoritmo de *hashing* utilizado é o MD5.[4]

De forma a proteger a privacidade dos utilizadores, o DCC apenas guarda as assinaturas do corpo das mensagens, alguns cabeçalhos e os endereços dos emissores, ou seja, é dada mais importância à privacidade dos receptores do que à dos emissores, uma vez que o envio de *email* em massa não é de todo privado.[4] Muitas vezes, algumas organizações que utilizam o DCC, preferem ter os próprios servidores de DCC em vez de utilizarem os servidores públicos, de forma a garantir mais privacidade.

2.4 Reputação

Continuando a analogia utilizada anteriormente para explicar os sistemas de *fingerprinting*, podemos também afirmar que muito antes de ser utilizado na informática, o conceito de reputação já era algo profundamente incutido na sociedade. A rep-

utação é um conceito que tipicamente está associado a uma entidade[7] e representa a opinião que os participantes de uma comunidade[8] têm sobre essa entidade. Ou seja, sendo uma comunidade algo em que várias entidades participam, tipicamente para ganhar algo, e sendo para isso necessário dar algo em troca à comunidade, a reputação é a métrica que permite aos participantes da comunidade saber em que entidades confiar mais ou menos. Em todas interacções efectuadas dentro de uma comunidade as entidades têm sempre em conta interacções passadas para avaliar como devem proceder na interacção actual. Assim, para que uma entidade possa manter uma boa reputação, apenas tem que seguir as regras impostas pela comunidade em que participa. A reputação é algo extremamente importante numa comunidade, uma vez que funciona como incentivo para que os participantes se comportem bem nas suas interacções.[13] Mesmo assim, certas entidades que são novas numa comunidade podem tentar efectuar interacções em que obtêm ganhos sem dar nada em troca à comunidade. Quando uma entidade X chega a uma comunidade pela primeira vez, as outras entidades não sabem qual é a reputação da entidade X uma vez que não têm informação passada sobre esta entidade. A entidade X pode tentar aproveitar isto para ganhar algo sem dar nada em troca à comunidade, e abandonar a comunidade logo a seguir. Isto faz com que a reputação entre estranhos seja muito mais difícil de se construir.[13]

O aparecimento da Internet deu origem ao aparecimento de muitas mais comunidades, e com isto, muitas mais interacções[9]. A Internet funciona numa escala muito maior que uma cidade, ou qualquer "comunidade *offline*", e permite anonimato em grande parte das interacções que são efectuadas. Isto é visto como uma tentação muito grande para que algumas entidades tentem ganhar algo de certas interacções sem dar a sua parte em troca, uma vez que de futuro ninguém saberá quem participou naquela interacção.[13]

O aparecimento dos sistemas de reputação é portanto um resultado inevitável do aparecimento da Internet. Os sistemas de reputação agregam informação sobre todos os participantes de uma comunidade, informação esta que é construída com base no histórico de interacções entre as várias entidades. Esta informação é usada para calcular os valores de reputação que ficam associados às entidades, e com base no valor de reputação de determinada entidade X, as outras entidades escolhem se é ou não seguro interagir com a entidade X. Os sistemas de reputação são cada vez mais adoptados em "comunidades *online*", como alguns fóruns, redes de partilha de ficheiros (P2P), *sites* de compra e venda de produtos (lojas *online*, leilões *online*),

[7] Uma entidade neste contexto pode ser um indivíduo, um grupo de indivíduos, uma organização, etc.

[8] Uma comunidade pode ser por exemplo uma cidade/localidade, grupos religiosos, escolas, a sociedade em geral, etc.

[9] Podemos mesmo considerar a Internet como uma única comunidade que engloba várias "sub-comunidades".

etc, que são comunidades que requerem justiça nas interacções entre os participantes, para garantir o seu bom funcionamento.

Mais recentemente, começou-se a tornar bastante popular a adopção de sistemas de reputação na filtragem de *email*, para detectar possíveis emissores de *spam*. As comunidades usadas como exemplo no parágrafo anterior têm todas a particularidade das entidades serem obrigadas a registar-se para poderem participar na comunidade. O envio de *emails* faz uso do protocolo SMTP que não requer autenticação por parte de quem envia, o que permite ao emissor forjar o seu próprio endereço de *email* com bastante facilidade[10] e fazendo assim com que seja muito difícil responsabilizá-lo. Existem soluções como o SPF (Sender Policy Framework), que verifica num *DNS Record* se o endereço IP de onde foi enviado o *email* é um dos que corresponde ao domínio do endereço de *mail* do emissor[15], ou o DKIM (DomainKeys Identified Mail), que utiliza um esquema de chave pública para provar que o *email* é de facto proveniente do servidor de *mail* correspondente ao domínio do endereço de *email*[15]. No entanto, para que estas soluções funcionem, é preciso que os receptores verifiquem se todos os *emails* que recebem estão devidamente autenticados, além de que nada impede os *spammers* de criarem o seu próprio servidor de *mail* e autenticarem o respectivo domínio. Outra possível forma de associar os valores de reputação, sem ser com o endereços de *email* ou os respectivos domínios, é associa-los com os endereços IP dos emissores. No entanto, tendo em conta que os IPs não estão sempre associados às mesmas entidades, os valores de reputação acabam por ter um "prazo de validade".

Apesar destes problemas, existem sistemas de reputação, usados na filtragem de *emails,* de todos os tipos mencionados anteriormente. Mesmo que qualquer um deles esteja longe de ser infalível, estes sistemas conseguem ser bastante eficazes e ajudam a reduzir drasticamente a quantidade de *spam* recebida. De seguida são apresentados alguns destes sistemas.

2.4.1 *Gmail Sender Reputation*

O *Gmail* é o serviço de *Webmail* da *Google* que se está a tornar cada vez mais popular, com o número de utilizadores a crescer a um ritmo elevado. Entre os vários mecanismos utilizados pela *Google* para fazer a filtragem de *emails*, está o sistema de reputação do *Gmail*.

Para abordar este problema, a *Google* criou a sua própria definição de *spam*. Em vez de ver o *spam* como *emails* não solicitados e enviados em massa, a *Google* decidiu considerar *spam* como sendo apenas *emails* indesejados[11].[14] Isto é válido no *Gmail*

[10] O emissor apenas tem que colocar o endereço de *email* que quiser nos cabeçalhos "*From*", "*Return-path*", entre outros.

[11] *Email* indesejado é diferente de *email* não solicitado, uma vez que é possível que um *email* não solicitado seja do interesse do receptor (por exemplo, um *email* cujo emissor é o departamento de recursos humanos de uma empresa em processo de recrutamento de colaboradores).

uma vez que a informação utilizada para calcular a reputação de um emissor, consiste principalmente no número de vezes que os utilizadores reportaram os *emails* daquele emissor como *spam* ou como *"not spam"*. Ou seja, do ponto de vista do sistema de reputação do *Gmail* são os utilizadores que decidem que *emails* são desejados e quais são indesejados.[14]

Os valores de reputação ficam associados aos domínios dos endereços dos emissores[12]. Tendo em conta as limitações do SMTP, a atribuição de valores de reputação aos domínios dos emissores não seria possível, uma vez que não há garantia que um *email* tenha sido enviado a partir do endereço que aparece no cabeçalho. Para resolver este problema, a *Google* tenta autenticar todos os domínios com SPF e DKIM/*DomainKeys*.[14] O SPF utilizado no *Gmail* consiste numa versão do SPF normal, com mais algumas características que foram adicionadas para poder autenticar mais domínios, e funciona da seguinte forma[14]:

1. Primeiro tenta-se autenticar o domínio com SPF normal, verificando num *DNS Record* se o endereço IP do emissor é um dos que pode enviar *emails* a partir do domínio usado pelo emissor.

2. Se o emissor não usar SPF, o *Gmail* utiliza um "*Best-guess SPF*", que assume um domínio como estando autenticado se o endereço IP do emissor estiver na mesma gama de IPs que os *records* A ou MX, ou se o nome no *PTR record* do endereço IP do emissor corresponder ao domínio usado.

3. Se nem o SPF normal nem o "*Best-guess SPF*" resultarem, o domínio ainda pode ser autenticado se este for um subdomínio do que aparece no *PTR Record*[13].

As *DomainKeys* também são usadas pelo *Gmail* para autenticar os domínios. Todos os *emails* que usam autenticação baseada em *DomainKeys* incluem um cabeçalho que vai cifrado com a chave privada do domínio do emissor. Este cabeçalho é posteriormente decifrado pelo receptor com a chave pública do domínio do emissor, verificando assim que emissor faz parte do domínio que diz ser. Muitos dos *emails* enviados para o *Gmail* são reencaminhados, e enquanto que estes *emails* são um problema para o SPF, uma vez que é difícil saber qual o IP que enviou o *email* originalmente, as *DomainKeys* autenticam estes domínios sem problemas.[14]

Para os domínios não autenticados, o sistema de reputação do *Gmail*, associa os valores de reputação aos endereços IP dos emissores.[14] Isto pode ser um problema para servidores de *email* que mudem para IPs com reputação comprometida, mas para este caso a *Google* apenas recomenda que os domínios sejam devidamente autenticados, para que a reputação possa ficar associada ao domínio e não ao endereço IP.

[12] Exemplo: o emissor *sender@domain.tld* faz parte do domínio *domain.tld*

[13] Exemplo: O domínio *subdomain.domain.tld* é autenticado se o *PTR Record* do endereço IP do emissor for *othersubdomain.domain.tld*

O cálculo dos valores de reputação é feito com base nos seguintes valores[14]:

- *autospam*: Quantos *emails* deste emissor foram automaticamente para a pasta de *spam*.
- *autononspam*: Quantos *emails* deste emissor foram automaticamente para a *inbox*.
- *manualspam*: Quantos *emails* deste emissor foram reportados como *spam* por utilizadores.
- *manualnonspam*: Quantos *emails* deste emissor foram reportados como "*not spam*" por utilizadores.

Estas variáveis são actualizadas à medida que chegam novos *emails* e à medida que os utilizadores os reclassificam ao utilizar os botões "*Report spam*" e "*Report not spam*". Para calcular a reputação é utilizada a seguinte fórmula.[14]

$$totalmsgs = autospam + autononspam \tag{2.1}$$

$$manualnonspam2 = min(autospam, manualnonspam) \tag{2.2}$$

$$manualspam2 = min(autononspam, manualspam) \tag{2.3}$$

$$good = autononspam + manualnonspam2 - manualspam2 \tag{2.4}$$

$$reputation = \frac{100 \cdot good}{totalmsgs} \tag{2.5}$$

Para evitar dividir por 0, quando a variável *totalmsgs* é igual a 0, não existe reputação para o emissor em questão. A reputação resultante desta fórmula é um valor entre 0 e 100 que representa a probabilidade dos *emails* de um emissor serem legítimos (*ham*).[14]

2.4.2 *DCC Reputations*

O *DCC Reputations* é uma componente do DCC (*Distributed Checksum Clearinghouses*) que foi mencionado mais atrás na secção de *fingerprinting*, e é responsável por calcular e guardar os valores de reputação dos emissores. Para poder explicar o *DCC Reputations*, é importante relembrar que a noção que o DCC tem de *spam* está limitada ao facto dos *emails* serem ou não enviados em massa (*bulk mail*), ignorando o facto dos *emails* serem solicitados ou não solicitados. Isto obriga a que os clientes

do DCC adicionem emissores de *bulk mail* solicitado a uma *whitelist* para que estes *emails* não sejam rejeitados.

Para calcular a reputação dos emissores, o *DCC Reputations* utiliza uma heurística bastante simples, que se limita a fazer um rácio entre o número de mensagens enviadas em massa e o número total de mensagens enviadas. O valor da reputação de um determinado emissor não é mais que a percentagem de mensagens enviadas em massa por ele[3]:

$$reputation = \frac{bulkmsgs}{totalmsgs}.100 \tag{2.6}$$

Se um emissor tiver por exemplo a reputação de 50, significa que 50% dos *emails* enviados por este emissor foram consideradas *bulk mail*.

O método usado para saber quantas mensagens determinado emissor enviou em massa, consiste em verificar a contagem de todas as assinaturas de *emails* (*checksums*) enviadas por aquele emissor, e aquelas que foram enviadas mais do que um certo número de vezes são consideradas *bulk mail* e entram para a contagem da variável *bulkmsgs*. As mensagens rejeitadas devido a má reputação do emissor, passam a ser guardadas com uma contagem de "*MANY*", servindo isto para que esta mensagem seja também rejeitada caso seja enviada por outro emissor (ou outro IP detido pelo mesmo emissor).[3]

O identificador ao qual fica associado o valor de reputação é o endereço IP do emissor. Uma vez que os endereços IP não estão sempre associados ao mesmo emissor, as reputações são expiradas uma semana a 30 dias depois de ter sido enviada a última mensagem de *bulk mail* reportada por um cliente do *DCC Reputations*.[3]

2.5 Discussão

As tabelas 2.1 e 2.2 são uma síntese das características dos sistemas de *fingerprinting* e reputação mencionados anteriormente. Uma vez que não existe uma única definição de spam, cada sistema tem que escolher a sua própria definição para saber o que vai ser filtrado. É por esta razão que ambas as tabelas contêm uma coluna onde é referida a "noção de *spam*" de cada sistema. No caso do *Razor* e do *Pyzor* a documentação não refere qual é a definição utilizada, mas pode-se inferir que é *email* indesejado, uma vez que os *reports* são feitos manualmente.

Podemos ver na tabela 2.1 que o *Razor* e o *Pyzor* são bastante semelhantes. O *Pyzor* sendo originalmente uma implementação do *Razor* noutra linguagem (*Python*), apresenta as mesmas características, com excepção da rede de confiança, possibilidade de revogação de assinaturas e os múltiplos *engines* (e os respectivos tipos de assinaturas). O DCC tem uma noção de *spam* diferente das do *Razor* e do *Pyzor*, fazendo com que as outras características sejam bastante distintas. Para o DCC, *spam* é

	Noção de *spam*	**Assinaturas**	***Reports***	***Checks***	**Informação mais relevante guardada**	**Outras características**
Vipuls's Razor	*Emails* indesejados.	Vários tipos de assinaturas sendo geradas de forma diferente de acordo com o *engine* usado: VR1 - compatível com *Razor V1*; VR2 - Texto inteiro; VR3 - N-Gramas (*fuzzy*); VR4 - Assinaturas efémeras (*fuzzy*); algoritmo de *hashing* é o SHA1.	Manuais e só para utilizadores registados.	Tipicamente configura-se o cliente de *mail* para fazer a verificação automaticamente.	Assinaturas e o corpo das mensagens de *mails* considerados *spam* por um número suficiente de utilizadores (este último é usado no algoritmo das assinaturas efémeras).	Preprocessador de texto, possibilidade de revogação de assinaturas (manualmente), múltiplos *engines* que suportam vários tipos de assinaturas, rede de confiança (TeS).
Pyzor	*Emails* indesejados.	Geradas através de um algoritmo que ignora partes do texto da mensagem. algoritmo de *hashing* é o SHA1.	Manuais.	Tipicamente configura-se o cliente de *mail* para fazer a verificação automaticamente.	Apenas as assinaturas de *mail* considerado *spam* por um número suficiente de utilizadores.	Preprocessador de texto
DCC	*Emails* enviados em massa (*bulk mail*)	Para serem geradas são ignoradas partes do texto que não alteram o significado da mensagem (*fuzzy*). algoritmo de *hashing* é o MD5.	N/A Os *checks* no DCC funcionam como *reports*.	Tipicamente configura-se o cliente de *mail* para fazer a verificação automaticamente. Estes *checks* funcionam também como *reports*.	Todas as assinaturas (incluindo as de *mail* legítimo) e os endereços dos emissores.	Os utilizadores devem manter uma *whitelist* que contenha os emissores de *bulk mail* cujos *emails* não querem ver rejeitados.

Tabela 2.1: Comparação entre os sistemas de *fingerprinting* analisados.

qualquer mensagem que seja enviada em massa, não havendo a distinção entre *mail* solicitado e não solicitado. Isto faz com que não seja necessária a existência de *reports*. Em vez disto, um utilizador apenas tem que verificar (*check*) para cada *mail*, se a respectiva assinatura já consta da base de dados, e neste processo o contador da assinatura é automaticamente incrementado. É por esta razão que os *checks* funcionam como *reports* no DCC. A informação guardada pelo DCC inclui os endereços dos emissores, pois isto será útil para o cálculo da reputação dos mesmos.

Na tabela 2.2 podemos ver as características de dois sistemas de reputação muito diferentes. Tal como a componente de *fingerprinting*, também a componente de reputação do DCC vê o *spam* como *mails* enviados em massa, o que explica o facto do valor de reputação de cada emissor não ser mais que a percentagem de *bulk mail*

	Noção de *spam*	Identificador utilizado	Valor de reputação	Métricas utilizadas
Gmail Sender Reputation	*Emails* indesejados.	Domínio do emissor para domínios autenticados com SPF e/ou *DomainKeys* e endereço IP para domínios não autenticados.	O valor de reputação representa a probabilidade dos *emails* de um emissor serem legítimas.	Nº de *mails* automaticamente na *inbox*; Nº de *mails* automaticamente na pasta *spam*; Nº de *mails* manualmente reportados como "*not spam*"; Nº de *mails* manualmente reportados como *spam*;
DCC Reputations	*Emails* enviados em massa (*bulkmail*).	Endereço IP dos emissores.	Percentagem de mensagens enviadas em massa por um determinado emissor.	Nº total de *bulk mail* enviado pelo emissor; Nº total de *mails* enviados pelo emissor;

Tabela 2.2: Comparação entre os sistemas de reputação analisados.

enviado. O identificador ao qual fica associado o valor de reputação também difere nestes dois sistemas. No caso do *Gmail*, é utilizado o domínio do emissor sempre que seja possível autentica-lo, enquanto que no DCC é sempre utilizado o endereço IP do emissor. Isto faz com que o sistema de reputação do DCC seja obrigado a expirar os IPs ao fim de algum tempo de inactividade, além de que a quantidade de informação guardada é muito maior, uma vez que tem que existir uma entrada para cada emissor (um domínio é quase sempre usado por vários emissores). Por outro lado, associar a reputação ao domínio do emissor, pode por vezes prejudicar ou favorecer injustamente outros emissores do mesmo domínio.

Capítulo 3

Arquitectura do *Spike*

O *Spike* é um sistema da *AnubisNetworks* que é essencialmente responsável por filtrar o máximo possível de *spam* sem ter que sujeitar os *emails* à pesada análise dos tradicionais filtros de análise de conteúdo. Entre os vários sistemas que fazem parte do *Spike*, destacam-se os sistemas de *fingerprinting* e de reputação, que são o alvo deste projecto.

Neste capítulo introduzem-se as componentes mais importantes do *Spike* de forma a melhor enquadrar o trabalho desenvolvido neste projecto. Nas secções 3.2 e 3.3 são apresentadas as arquitecturas actuais do *spike client* e do *spike aggregator*, que são as componentes que contêm os sistemas de *fingerprinting* e reputação abordados nesta tese.

3.1 Organização geral do sistema

Existem três componentes principais no *Spike*: *Spike Client*, *Spike Node* e *Spike Aggregator*.

O *Spike Client* é a componente que fica instalada no servidor de *mail* do receptor e que é responsável por intersectar os *emails* para gerar uma assinatura a partir do corpo da mensagem. Depois de processar a mensagem, o *Spike Client* envia um *spike packet* para o *Spike Node*, um pacote onde vão incluídos vários dados, nomeadamente a assinatura gerada e o endereço IP do emissor. Esta assinatura é verificada contra uma base de dados que contem todas as assinaturas de *mails* previamente considerados *spam*, e é feito um pedido DNS para o *Spike Aggregator* para verificar a reputação do endereço IP do emissor. Depois de verificada a assinatura e a reputação do IP do emissor, o *Spike Node* responde ao *Spike Client* com o resultado. Em caso da assinatura estar presente na base de dados do *Spike Node* e/ou o emissor ter má reputação, o *Spike Client* descarta o *email*.

Neste processo, o *Spike Node* também acumula os IPs e assinaturas que vão chegando nos pacotes enviados pelo *Spike Client* para periodicamente os enviar para o *Spike Aggregator*. No *Spike Aggregator* é onde é calculada e actualizada a reputação dos IPs enviados periodicamente pelo *Spike Node*.

Nos pontos seguintes, descreve-se em maior detalhe cada uma destas três componentes.

3.1.1 *Spike client*

Ao dar entrada no MTA do receptor, os *emails* são processados pelo *Spike Client* antes de serem sujeitos aos filtros de *mail* tradicionais. O *Spike Client* começa por extrair informação sobre o tipo de conteúdo que vai no *email*, assim como o tipo de *encoding* do texto (no caso do conteúdo ser texto). Esta informação é importante porque o *email* pode conter vários *emails* encapsulados, e nestas situações para saber como extrair a mensagem original (que se encontra dentro de um desses *emails*), é utilizada esta informação. Depois de extraído o texto da mensagem original, este vai ser preparado, para que possa ser processado pelo algoritmo que gera as assinaturas. A figura 3.1 mostra o fluxo dos *emails* que passam pelo *Spike Client*.

Após ter sido corrido o algoritmo que gera a assinatura, já existe toda a informação necessária para construir o *spike packet* que vai ser enviado para o *Spike Node*. A especificação do *spike packet* encontra-se no Anexo A. Depois de construído o *spike packet*, este é enviado para o *Spike Node* para verificar se a assinatura consta do catálogo de *spam* conhecido pelo *Spike*, que por sua vez responde ao *Spike Client* com o resultado desta verificação.

As assinaturas geradas pelo algoritmo do *Spike*, têm algumas particularidades. O algoritmo funciona de forma a que dois *emails* parecidos dêem origem a duas assinaturas parecidas. Assim, a assinatura resultante é uma assinatura do tipo MD5 (32 caracteres hexadecimais), mas que pode ser dividida em quatro partes de oito caracteres. Por exemplo, se duas mensagens forem quase iguais, excepto nalgumas palavras mesmo no final do texto, é provável que as assinaturas resultantes desses textos sejam iguais excepto nos últimos oito caracteres. Este algoritmo é descrito em mais detalhe no anexo B.2, mas serve isto para explicar que o resultado da verificação de uma assinatura no *Spike Node* pode tanto pode ser um *match* exacto, como também pode ser um *match* parcial no caso de nem todas as quatro partes da assinatura serem iguais.

3.1.2 *Spike Node*

O *Spike Node* é onde está situada a base de dados que guarda as assinaturas de *emails* que o *Spike* conhece como sendo *spam*. Sempre que o *Spike Node* recebe um *spike*

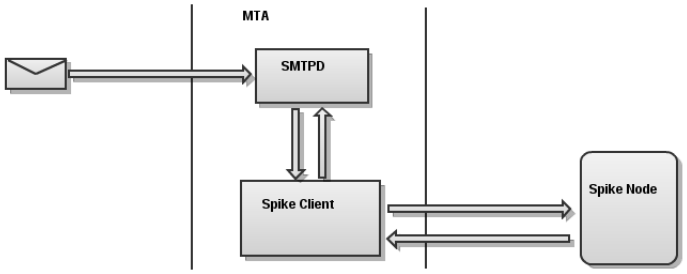

Figura 3.1: *Spike client*

packet de um dos *Spike Clients*, verifica se assinatura existe na base de dados, e envia um pedido DNS para o *Spike Aggregator* para verificar a reputação do IP do emissor, que também vai incluído no pacote. Na base de dados também são guardados todos os URIs que apareceram em *emails* considerados *spam*. Isto serve para verificar se os URIs que vão incluídos no pacote também se encontram na base de dados, caso em que a mensagem deverá automaticamente ser considerada *spam*. O *Spike Node* responde depois ao cliente com o resultado da verificação da assinatura, URIs e/ou da reputação do IP.

Uma vez que não existe um único *Spike Node*, é necessário que periodicamente estes sincronizem as respectivas bases de dados, para que todos os *Spike Clients* tenham acesso ao catálogo completo de assinaturas. Além disto, à medida que os *Spike Nodes* recebem pedidos de verificação dos respectivos *Spike Clients*, vão acumulando um conjunto de entradas do tipo <ip, assinatura, *ham*/*spam*>, que periodicamente são enviadas para o *Spike Aggregator*, para que seja calculada e actualizada a reputação dos IPs dos emissores.

A figura 3.2 demonstra o funcionamento de todo este processo.

3.1.3 *Spike Aggregator*

O *Spike Aggregator* faz exactamente o que diz o nome, ou seja, agrega informação sobre os emissores e o seu comportamento no que diz respeito ao envio de *emails*. Como já foi mencionado acima, periodicamente o *Spike Node* envia um conjunto de itens do tipo <IP, assinatura, *ham*/*spam*> para o *Spike Aggregator*, para que este possa calcular a reputação dos IPs que vão nestes itens. O *Spike Aggregator* calcula a reputação do IP com base no facto da respectiva assinatura estar marcada como *ham* ou *spam* e com base no histórico de envio de *emails* desse mesmo IP. Depois de processados os itens, o *Spike Aggregator* actualiza a base de dados de reputação com os novos valores de reputação. Neste processo também se guarda na base de

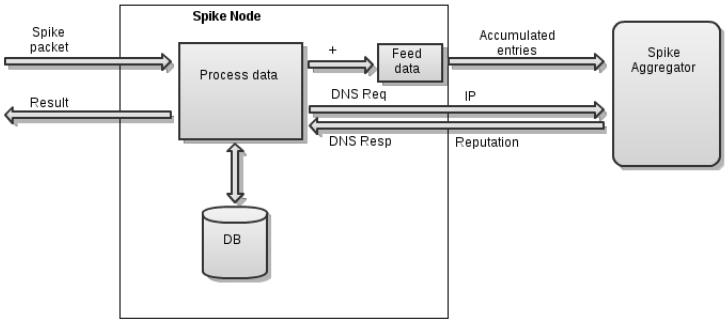

Figura 3.2: *Spike Node*

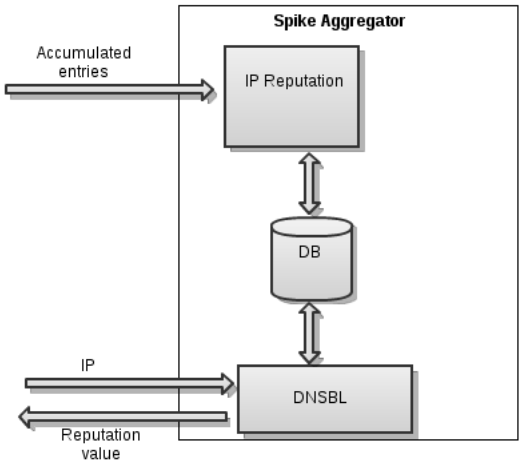

Figura 3.3: *Spike Aggregator*

dados informação extra sobre os IPs (*autonomous system number*, *PTR Record*, gama de IPs a que pertence, etc), as assinaturas dos *emails* enviados pelos IPs, e *botnets* detectadas no processo.

Existe também uma DNSBL no *Spike Aggregator*, que recebe pedidos DNS que contêm o endereço IP de um emissor, provenientes dos *Spike Nodes* (ou mesmo de *end-users*), e que responde com o respectivo valor de reputação. A figura 3.3 ilustra a descrição anterior.

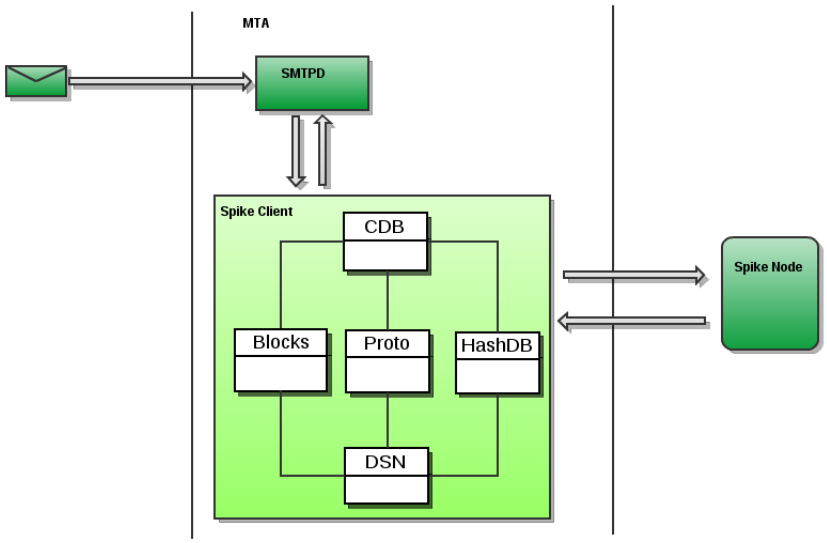

Figura 3.4: Organização da arquitectura do actual *spike client*

3.2 Arquitectura do *spike client* actual

O actual *Spike client* consiste num conjunto de módulos de *Perl*, que extraem o texto, geram a assinatura e constroem o pacote que é enviado para o *Spike Node*. Este conjunto de módulos constituem uma biblioteca denominada *libspike*, que é chamada para cada *email* que dá entrada no MTA. Por cada *email* que é recebido como entrada, a *libspike* envia o pacote com a assinatura da mensagem para o *Spike Node*, e devolve ao SMTP *daemon* a resposta do *Spike Node* sobre o que fazer com o *email*.

A figura 3.4 ilustra o *spike client* actual e as funções de cada módulo são as seguintes:

Proto Começa por fazer a captura da mensagem original e tratamento do texto da mensagem. Para a primeira parte, são percorridas as várias partes MIME do *email*, e é extraído o texto da mensagem original. Para saber em que partes MIME está texto da mensagem original, é necessário olhar ao tipo de conteúdo de cada parte. A segunda parte (tratamento do texto) aplica-se quando parte do texto (ou a sua totalidade) está em HTML e/ou codificado em *quoted-printable* ou base64, e serve para o transformar em texto limpo (*plain text*).

Depois de corridos os outros módulos da *libspike*, o módulo *Proto* é ainda responsável por criar o *spike packet* mencionado na secção anterior, e por o enviar para o *Spike Node*.

Blocks Faz a computação de um *template* da mensagem, que consiste num conjunto de letras em que cada letra representa uma característica da mensagem, como *line feads*, *carrige returns*, secções de HTML, URIs, endereços de *mail*, secções de texto normal, parágrafos e secções de texto que fujam à normalidade.

HashDB Depois de extraído o texto e calculado o *template* da mensagem, este módulo gera uma assinatura a partir do texto preprocessado pelo módulo Proto.

CDB Chama os três módulos anteriores para calcular uma *hash* da mensagem original contida no *email*.

DSN No caso do *email* ser um DSN (*Delivery Status Notification*), este módulo chama os três primeiros módulos mencionados, mas desta vez para calcular uma *hash* do texto das partes MIME que encapsulam as partes que contêm a mensagem original, mais propriamente as partes que referem NDRs (*Non Delivery Reports*).

3.3 Arquitectura do *spike aggregator* actual

O *Spike aggregator* é a componente do *Spike* que contem o sistema de reputação. O sistema de reputação é constituído essencialmente por três componentes: obtenção de informação sobre os IPs dos emissores, cálculo da reputação desses IPs e construção de *clusters* de *zombies* (ou *botnets*). A informação sobre os IPs inclui asn (*autonomous system number*), código de país, rede (*bgp prefix*) e PTR *record*. Este último entra para os cálculos dos valores reputação e os outros servem para fins estatísticos e para eventuais funcionalidades que sejam adicionadas. Os *clusters* de *zombies* são mais uma forma de saber que emissores devem ser barrados. Se o IP de um emissor fizer parte de um *cluster*, os seus *emails* devem ser automaticamente considerados *spam*.

O actual *aggregator* está dividido em duas camadas principais: a camada de persistência (base de dados) e a camada aplicacional que gere e processa os dados. Além destas duas camadas existem ainda alguns *scripts* auxiliares que efectuam actualizações periódicas na base de dados.

A base de dados tem uma estrutura bastante simples, não havendo qualquer relação entre as tabelas. A figura 3.5 sumariza as tabelas existentes na base de dados, sendo dada de seguida uma breve explicação sobre as mesmas.

Para o cálculo dos valores de reputação, a base de dados tem quatro tabelas, uma onde são guardados os valores de reputação actualizados (tabela *reputation*) e outras três que servem de apoio ao cálculo dos mesmos. A tabela *helpers* guarda o número de mensagens *ham* enviadas pelo maior emissor de *ham* e o número de dias em que isto aconteceu, informação que é utilizada na heurística de reputação. A tabela *t_today*

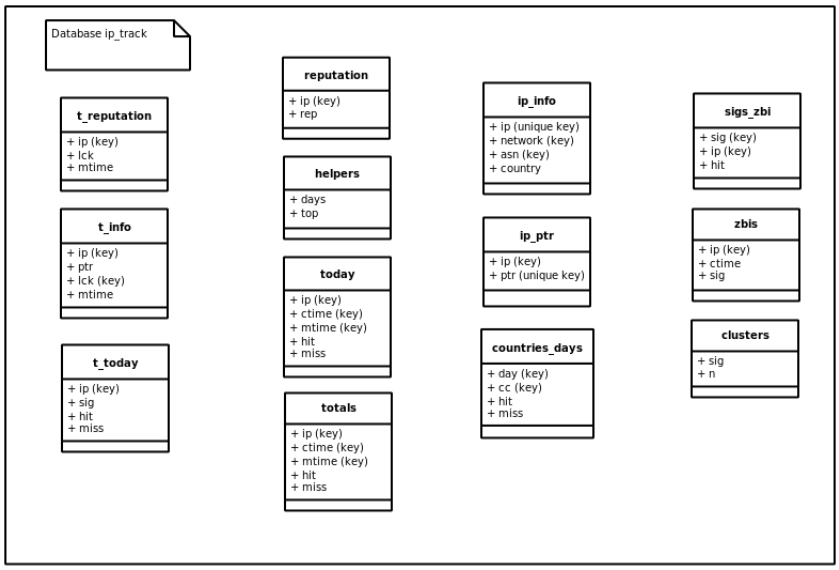

Figura 3.5: Base de dados do actual *spike aggregator*

serve de apoio à população da tabela *today* que guarda o número de *ham* e *spam* enviado por cada emissor no dia corrente. A tabela *totals* é diariamente actualizada para guardar o total de *ham* e *spam* enviado por IP, durante os dias em que esse IP esteve activo[1].

Para guardar a informação sobre os IPs são usadas as tabelas *ip_info*, *t_info* e *ip_ptr*. A tabela *t_info* serve de tabela auxiliar para popular a tabela *ip_ptr* com os PTR *records* dos IPs, e a tabela *ip_info* guarda a restante informação sobre os IPs.

A descoberta de *clusters* de *zombies* é feita com as tabelas *sigs_zbi*, *zbis* e *clusters*. Para fazer a descoberta de *zombies*, é necessário descobrir vários IPs que tenham enviado os mesmos *emails* (ou seja, as mesmas assinaturas). Para fazer isto, a tabela *sigs_zbi* contem pares <IP, assinatura> que permitem saber exactamente que assinaturas é que cada IP enviou, e um campo que diz se essa assinatura é ou não de um *email spam*. A tabela *zbis* guarda IPs que enviaram assinaturas iguais e as respectivas assinaturas (retirados da tabela *sigs_zbi*).

A camada aplicacional está dividida em quatro módulos (implementados em PERL): um módulo de carregamento de novas entradas, um para o cálculo e actualização da reputação, um para actualização da informação dos IPs e um para gestão e actualização dos *clusters* de *zombies*.

O módulo de carregamento (*ipload*) é responsável apenas por receber ficheiros com entradas do tipo <IP, assinatura, ham/spam> e colocá-las nas tabelas *t_today*,

[1]Considera-se que um IP está activo se não estiver sem enviar *emails* por mais de n dias. Este valor é ajustável.

t_reputation e *sigs_zbi*. Além disto este módulo também verifica quais dos IPs que vêm nas entradas ainda não se encontram na tabela *ip_ptr* e coloca-os na tabela *t_info*. Posteriormente os outros módulos irão fazer as actualizações necessárias partindo dos dados das tabelas *t_today*, *t_reputation*, *t_info* e *sigs_zbi*.

Para cada IP guardado na tabela *t_info*, o módulo de gestão de informação dos IPs (*ipinfo*) tenta resolve-lo para o seu PTR *record* e obtém o seu asn, prefixo de rede e código de país. O PTR é guardado na tabela *ip_ptr* e a restante informação na tabela *ip_info*. Todos os IPs actualizados desta forma, são por fim apagados da tabela *t_info*.

O módulo de reputação (*reputation*) começa por retirar da tabela *t_reputation* os IPs recém-chegados. Para cada um destes é calculado um valor de reputação através de uma função heurística que tem em conta o histórico de envio de *ham* e *spam* desse IP (tabelas *today* e *totals*), o seu PTR e tempo activo. Os IPs são depois inseridos ou actualizados na tabela *reputation* com os novos valores de reputação e são apagado na tabela *t_reputation*.

Para finalizar, o módulo de gestão de *clusters* retira todas as assinaturas que apareçam mais de 50 vezes na tabela *sigs_zbi* e que estejam classificadas como *spam* (de notar que esta tabela guarda pares <ip, assinatura> como chave primária, pelo que assinaturas que apareçam repetidas estarão certamente associadas a IPs diferentes). Para cada assinatura destas, o módulo obtém todos os IPs associados a ela, mais uma vez a partir da tabela *sigs_zbi*. Cada um destes IPs é de seguida inserido ou actualizado na tabela *zbis*, excepto os que não tenham reputação suficientemente má.

Existem ainda alguns *scripts* que efectuam actualizações periódicas na base de dados. Estas actualizações periódicas acontecem por exemplo para os valores de reputação (a descoberta dos PTRs é muito demorada e quando a reputação é calculada pela primeira vez para um IP, este pode ainda não ter um PTR *record* associado), para somar o conteúdo da tabela *today* à tabela *totals* diariamente e para expirar IPs e *clusters* inactivos. Estes *scripts* são particularmente importantes para contemplar o facto dos IPs não estarem sempre associados à mesma entidade, podendo hoje um IP estar associado a uma entidade mal intencionada e amanhã estar associado a uma entidade que pretende enviar *emails* genuínos.

Capítulo 4

Trabalho desenvolvido

Neste capítulo descreve-se detalhadamente o trabalho desenvolvido neste projecto. Na secção 4.2 descreve-se a organização da arquitectura do novo *spike client*. A secção 4.3 apresenta a organização da arquitectura do novo *spike aggregator*.

4.1 Introdução

Quando a *AnubisNetworks* iniciou a sua actividade, começou por utilizar apenas os filtros de conteúdos tradicionais na sua solução. No entanto rapidamente se notou que estes não tinham capacidade para processar todos os *emails* suficientemente rápido, pelo que apareceu a necessidade de criar o *MailSpike*. O *MailSpike* foi assim construído com pouco planeamento, mas resultando numa solução que conseguiu com sucesso resolver o problema de capacidade insuficiente de processamento. Sendo esta uma solução distribuída para clientes que optam por partilhar a informação sobre os *emails* que recebem, e havendo clientes com cada vez mais *emails* para processar, a carga de *emails* que são processados pelo *Spike* e a necessidade de acrescentar novas funcionalidades fazem com que a solução inicial se comece a tornar obsoleta. O *Spike* está portanto a ser completamente redesenhado para resolver estes problemas, num trabalho conjunto da empresa, sendo o âmbito do trabalho desenvolvido para esta tese o redesenho e implementação dos novos *spike client* e *spike aggregator*.

Para realizar este trabalho, começou-se por analisar detalhadamente a versão original do *Spike*, para compreender quais as funcionalidades existentes, e como é que funcionam. Por um lado este processo foi dificultado pela falta de documentação, obrigando assim à leitura exaustiva do código do sistema, por outro lado isto permitiu ganhar um nível de conhecimento pormenorizado do funcionamento do *Spike*. Depois de estudada a versão original do *Spike*, procedeu-se ao desenho e implementação dos vários sistemas que compõem o *Spike*, para mais tarde estes serem integrados numa

nova plataforma que os suportará. A integração com a plataforma ficou para o fim, para garantir que os novos sistemas têm as melhorias necessárias na capacidade de processamento e que as suas funcionalidades estão de acordo com o que é esperado.

Nas próximas secções são descritas as arquitecturas do novo *spike client* e do novo *spike aggregator*, seguidas dos algoritmos utilizados.

4.2 *Spike Client*

Para esta parte do projecto, os objectivos passam apenas por redesenhar a arquitectura do *Spike Client*, mas garantindo que o protocolo utilizado e o resultado final são os mesmos. O sistema resultante deve ser mais performante que o antigo, deve ser mais modular de forma a ser facilmente extensível e alterável, e deve corrigir os problemas do antigo.

4.2.1 Organização da arquitectura

O trabalho desenvolvido nesta parte do projecto assenta na refactorização total da arquitectura do *Spike Client*. O aumento de performance é um requisito que tipicamente é mais eficientemente concretizado criando o sistema de raiz. Assim, a *libspike* foi completamente redesenhada e adicionou-se uma API de gestão de conteúdos de *mail*, que facilita o acesso aos *emails* que passam pelo MTA e acelera o processamento dos mesmos.

A nova *libspike* inclui as mesmas funcionalidades que a actual, mas a estrutura da solução é diferente. A figura 4.1 representa a estrutura da nova solução e estes são os vários componentes que a constituem:

libanubisnet Para começar separou-se a lógica de comunicação, que permite ao *Spike Client* enviar os pacotes para o *Spike Node,* para uma biblioteca à parte. Isto serve principalmente para isolar a lógica de comunicação de forma a que se houver necessidade de fazer alguma alteração nesta componente o impacto restringe-se a esta biblioteca. A biblioteca contem um conjunto de classes, cada uma referente ao tipo de transporte a ser utilizado na comunicação. No caso da *libspike*, é utilizado o protocolo UDP para enviar os pacotes para o *Spike Node*, mas a vantagem é que mais tarde se se quiser utilizar outra forma de transporte (como por exemplo através de pedidos DNS), a alteração será bastante simplificada. Outras características desta biblioteca incluem *retries* e *timeouts* configuráveis e comunicação assíncrona. Esta biblioteca está implementada de forma suficientemente genérica para que possa ser utilizada por outras aplicações.

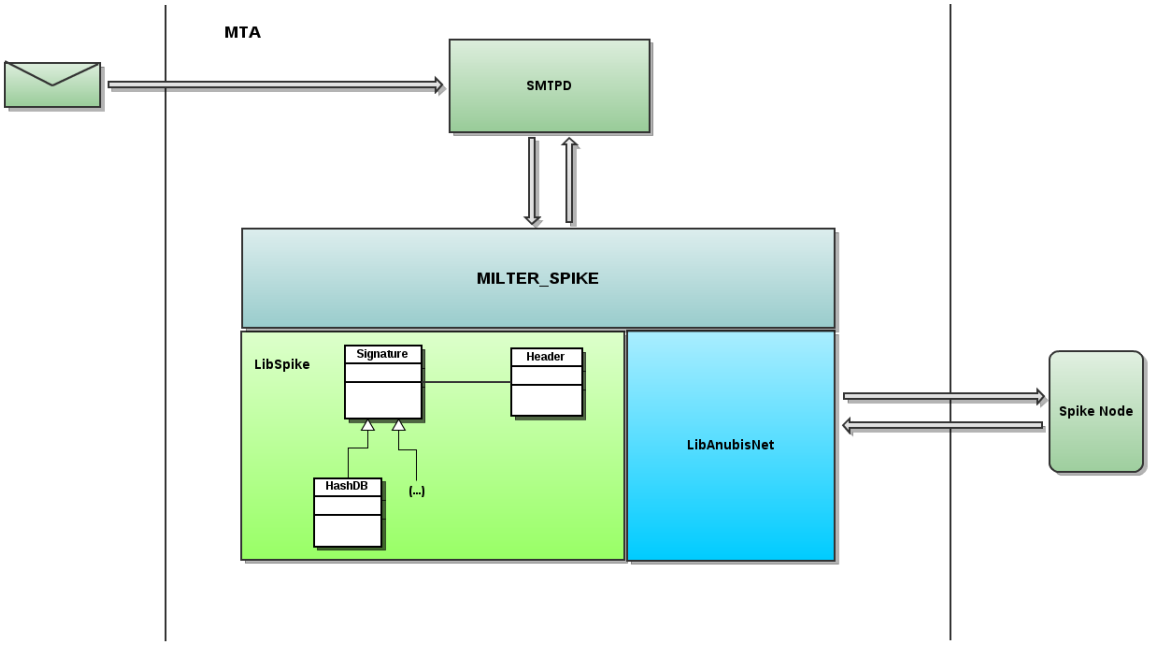

Figura 4.1: Reorganização do *Spike Client*

Signature Como já foi referido antes, está planeada na *Anubis* a adição de novos tipos de assinaturas ao *Spike*. Esta classe *Signature* é responsável por fazer a extracção e tratamento do texto da mensagem, computação do *template* da mensagem e envio do *Spike packet* para o *Spike Node*. Estas funcionalidades estão agrupadas nesta classe, pois é ela que servirá de base aos vários tipos de assinaturas, e estas funcionalidades são necessárias para qualquer tipo de assinatura que seja acrescentado.

Assim, esta classe começa por fazer o processamento das partes MIME para poder extrair a mensagem original do *email*. Os *standards* que definem como os *emails* devem ter as suas partes MIME (RFC 2045), são muito ambíguos e permitem muitos formatos diferentes, o que dificulta o processo de análise, pois é necessário contar com todas as variantes. Para resolver este problema, já existem algumas bibliotecas que implementam métodos para detectar os vários campos e conteúdos das partes MIME, entre as quais a *libmimetic* que é grátis e livre, e que torna o processo de extracção do texto pretendido mais fácil e fiável. Depois de extraída a mensagem original, esta é usada para se calcular o *template* que foi referido na explicação da arquitectura actual do *spike client*.

No final de todo o processo de geração das assinaturas, esta classe é também responsável pela construção e envio do *Spike packet* para o *Spike Node*. O pacote segue a mesma estrutura apresentada na explicação da arquitectura actual.

Header Esta classe é apenas responsável pela construção do cabeçalho do *Spike packet*. A classe *Signature* diz à classe *Header* que valores colocar nos vários campos, e no final a classe *Signature* apenas tem que juntar o cabeçalho ao *payload* que vai no pacote. Isto é particularmente vantajoso na possibilidade de haver alterações ao protocolo do *Spike*, pois neste caso grande parte das alterações iriam incidir apenas sobre esta classe.

HashDB A classe *HashDB* representa o tipo de assinaturas do *Spike* mencionadas até agora, e contem o algoritmo de geração destas assinaturas. Neste caso as assinaturas são baseadas no texto da mensagem original, mas outros tipos de assinaturas poderiam por exemplo ser utilizadas para identificar *emails* com imagens, sendo as assinaturas geradas a partir dessas imagens. Depois de gerada a assinatura, esta é colocada pela classe *Signature,* no *payload* do *Spike packet* juntamente com o tamanho do texto usado para gerar a assinatura, o peso do corpo da mensagem (um valor que resulta de uma variável utilizada no algoritmo de *hashing* explicado no anexo B.2) e o número de palavras no texto da mensagem original. Estes últimos valores podem ajudar a detectar colisões de assinaturas (mensagens completamente diferentes a gerarem a mesma assinatura). Duas mensagens que gerem a mesma assinatura e não tenham estes valores iguais, é quase de certeza uma colisão.

A outra grande alteração ao *Spike Client* foi a adição do *milter* (*mail filter*), uma API que gere os conteúdos dos *emails* à medida que estes passam pelo MTA[5], e que permite que outras aplicações tenham acesso aos dados contidos nos *emails*, para que possam ser analisados e/ou modificados durante a transacção SMTP. Quando integrado com a *libspike*, o *milter* lança uma *pool* de *threads* que ajudam a aumentar consideravelmente a capacidade de processamento da *libspike*, e permite que as assinaturas sejam geradas e verificadas assim que o *email* dá entrada no MTA e antes de serem sujeitos aos filtros de conteúdos. O *milter* inclui portanto uma biblioteca (*libmilter*) que contem um conjunto de *callbacks* que permitem manipular o *email* ao longo das etapas de transacção do SMTP[1].

A integração da *libspike* com a *libmilter* possibilitou a criação de um *daemon* (*milter_spike*) que intersecta os *emails* no MTA, e retira a informação referente à transacção e ao conteúdo dos *emails*, como o endereço IP do emissor, o comando HELO/EHLO, o endereço de origem (*envfrom*), o endereço de destino (*envrcpt*) e a própria mensagem. Parte desta informação (IP e mensagem) é passada à *libspike* para que seja gerada e verificada a assinatura da mensagem. Com a resposta obtida pelo *libspike* acerca da assinatura, o *milter* devolve um código que diz ao MTA para descartar a mensagem em caso da assinatura ser de uma mensagem *spam*.

[1]Por exemplo, o *callback* xxfi_helo permite extrair o campo HELO/EHLO do *email*, e se necessário, modificá-lo.

O *milter_spike* faz uso de um ficheiro de configuração onde são definidas várias opções que alteram o comportamento do *daemon*. Nas opções de filtragem pode-se definir o tamanho máximo das mensagens permitido, as gamas de endereços que devem ser consideradas confiáveis[2] e cabeçalhos que devem ser acrescentados. Existe também uma secção com opções referentes à *libspike*, onde se define o endereço do *Spike Node*, o porto de comunicação, a duração dos *timeouts* e o número de *retries* máximo em caso de *timeout*.

4.3 *Spike Aggregator*

Esta segunda fase do projecto assenta na refactorização completa do *Spike Aggregator*. O sistema actual tem vindo a crescer com o aparecimento de novas funcionalidades e foi ficando bastante confuso e complexo, sendo portanto necessária uma nova organização da sua estrutura. Mais uma vez, sendo o requisito de performance de elevada importância, uma vez que o sistema tem de estar preparado para receber e processar um grande número de entradas, decidiu-se recriar o sistema de raiz. Ao contrário da refactorização do *spike client* em que era importante manter o protocolo já existente, para o sistema de reputação houve mais liberdade para fazer alterações.

4.3.1 Modelo de dados

A base de dados do actual *spike aggregator* tinha o problema das tabelas terem muita informação replicada e de serem necessárias muitas tabelas auxiliares que servem para obter a informação que é realmente importante e que fica armazenada em apenas quatro tabelas (*reputation*, *ip_info*, *ip_ptr* e *zbis*). Para evitar isto, optou-se por implementar um desenho relacional para a base de dados do novo sistema. Isto leva não só a uma redução considerável no número de tabelas, mas também a uma organização que está mais de acordo com a lógica do sistema, tornando este modelo menos confuso e mais fácil de compreender. Para isto, foram identificadas três entidades que estão associadas a este sistema: os emissores, as mensagens enviadas por eles e os *clusters* de *zombies*.

O emissor é a entidade mais importante, uma vez que é a esta que os valores de reputação são atribuídos, e são estes valores que são questionados por quem recebe *emails*. Um emissor é identificado pelo endereço IP utilizado no envio do *email*. Além do valor de reputação, existem outros dados que estão associados ao emissor, que são usados no cálculo da sua reputação, como por exemplo a média, de *ham* e *spam* enviado até ao instante corrente. Existem ainda alguns dados que estão directamente

[2]Muitos *emails* incluem vários IPs nos cabeçalhos e para descobrir o endereço que corresponde ao emissor, é necessário encontrar o primeiro IP que não esteja em nenhuma destas gamas de endereços (que são normalmente gamas de redes locais).

associados ao endereço IP do emissor que correspondem à informação sobre os IPs mencionada na arquitectura do *aggregator* actual (PTR *record*, asn, prefixo de rede, país, e adicionalmente o registo e data de registo do IP).

As mensagens enviadas são representadas pelas suas assinaturas (geradas no *Spike Client*). Estas não têm qualquer influência no cálculo dos valores de reputação dos emissores, mas é importante que sejam armazenadas e que estejam associadas aos respectivos emissores, pois é a partir desta informação que os *clusters* de *zombies* são detectados. As assinaturas são guardadas na base de dados, divididas nas quatro partes que as constituem, de forma a tornar a detecção de *clusters* mais eficaz (este processo é explicado em maior detalhe na secção B.4). Uma vez que a tabela de assinaturas tem tendência a crescer rapidamente e que as assinaturas nela guardadas apenas são usadas na detecção de *clusters*, estas são apagadas ao fim de um número de dias configurável.

Os *clusters* de *zombies* são conjuntos de emissores em que dentro de cada conjunto todos os emissores têm em comum o facto de terem enviado o mesmo *email* (ou *emails* muito semelhantes). Os *clusters* são identificados por um ID, e têm associados a eles as assinaturas das mensagens enviadas pelos emissores que compõem o *cluster*. No *spike aggregator* actual um *cluster* podia ser identificado apenas pela assinatura da mensagem que todos os emissores desse *cluster* enviaram, pois era isso que permitia identificar o *cluster*. No novo *aggregator* isto não é possível porque considera-se que vários emissores enviam a mesma mensagem não só quando as assinaturas são iguais, mas também quando são parecidas. Isto significa que dentro de um *cluster*, as assinaturas associadas serão sempre semelhantes.

A figura 4.2 ilustra o novo modelo de dados utilizado nesta nova versão do *spike aggregator* e nela estão representadas as seis tabelas utilizadas pelo sistema. As entidades emissor, mensagem e *cluster* mencionadas acima são representadas pelas tabelas *Sender*, *Signature* e *Cluster* respectivamente. A tabela *Sender_Signature* faz a relação entre as entidades *Sender* e *Signature*, e permite saber exactamente que mensagens foram enviadas por que emissores e vice-versa (o atributo *count* diz quantas vezes um emissor enviou determinada mensagem). A tabela *Sender_Cluster* contem as relações entre os emissores e os *clusters* a que eles pertencem (aqui o atributo *count* representa o número de vezes que um IP contribuiu para um *cluster*). A informação referente aos IPs fica armazenada na tabela *Sender_Info*.

4.3.2 Modelo de reputação

A arquitectura da camada aplicacional do novo *spike aggregator* tem duas partes bastante distintas: uma camada (*libiprep*) que contem classes e métodos que processam os dados que resultam da informação acumulada no *Spike Node,* e três *daemons* que utilizam a *libiprep* para fazerem as actualizações necessárias. Nesta secção é explicada

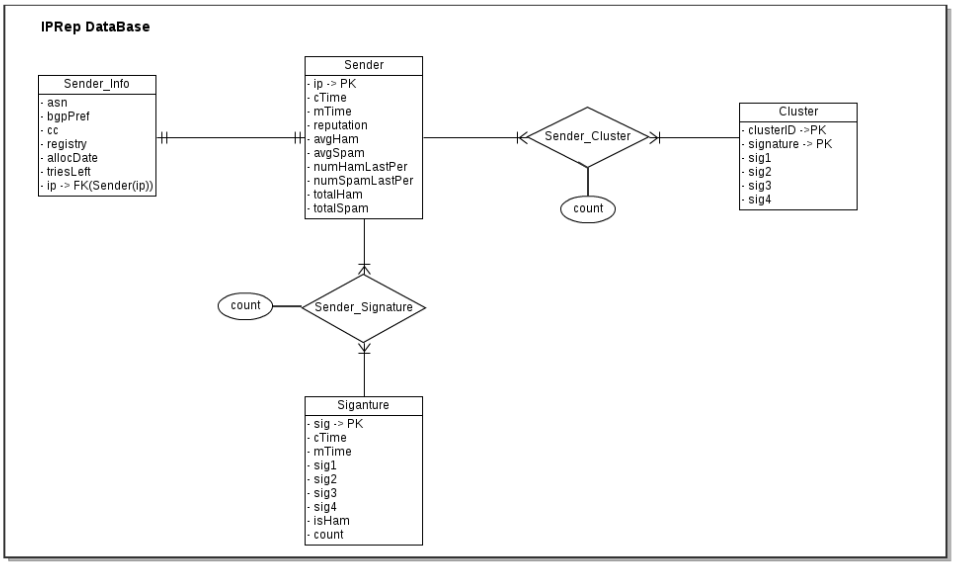

Figura 4.2: Modelo de dados do novo *spike aggregator*

o fluxo de informação que passa pelo *Spike Aggregator*, seguido da forma como está organizada a arquitectura da biblioteca de reputação, que estão representados nas figuras 4.3 e 4.4 respectivamente.

O *Spike aggregator* é um sistema que necessita de estar constantemente à espera de novas entradas (do tipo <ip, assinatura, *ham*/*spam*>), em que cada entrada representa o envio de um *email* (assinatura), classificado como *ham* ou *spam*, por parte de um emissor (ip). Ao receber novas entradas, o *iprep-daemon* (representado na figura 4.3), actualiza quatro *caches* distintas: a *cache* do emissor, a *cache* das mensagens, a *cache* da relação entre os emissores e as mensagens e a *cache* da informação dos IPs. Estas actualizações podem ser a adição de novas entradas nas *caches* ou actualização de alguns valores, no caso da entrada já existir na *cache,* como o valor de reputação no caso do emissor e incrementação de contadores de mensagens. A utilização destas *caches* para guardar temporariamente os dados que chegam, deve-se principalmente ao facto do acesso à base de dados ser muito mais demorado do que o acesso à memória. Assim, evita-se a utilização de tabelas temporárias para guardar as entradas antes destas serem processadas. As *caches* contém três operações essenciais: *load*, *save* e *expire*. Ao fazer *load* de uma entrada pode acontecer uma de três situações: a entrada existe na *cache* e esta é retornada; a entrada não existe na *cache*, mas existe na base de dados, sendo colocada na *cache* a entrada que está na base de dados; a entrada não existe nem na *cache* nem na base de dados, sendo construída uma nova entrada e colocada na *cache*. A operação de *save* procura na *cache* as

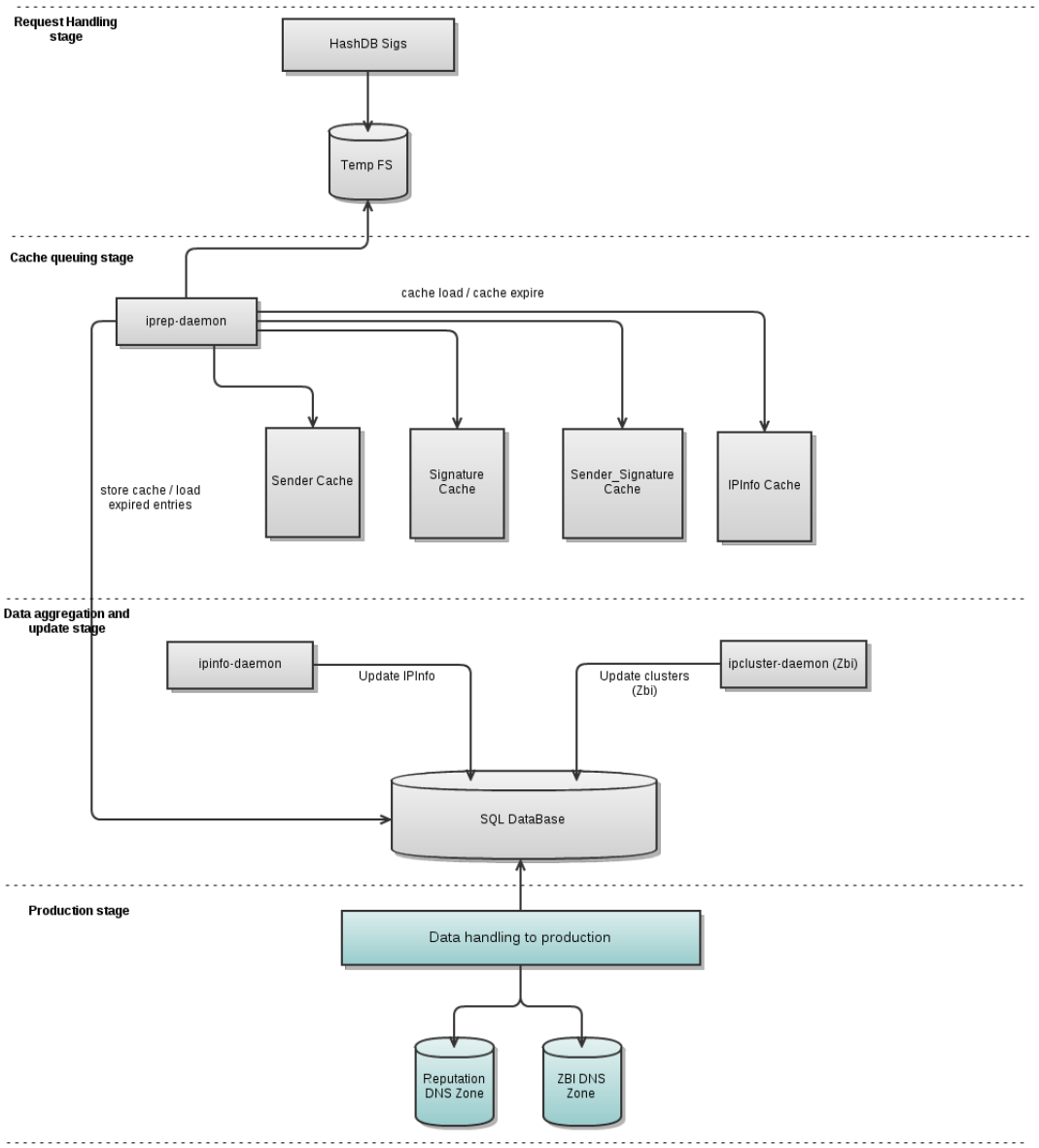

Figura 4.3: Fluxo de informação no *Spike Aggregator*

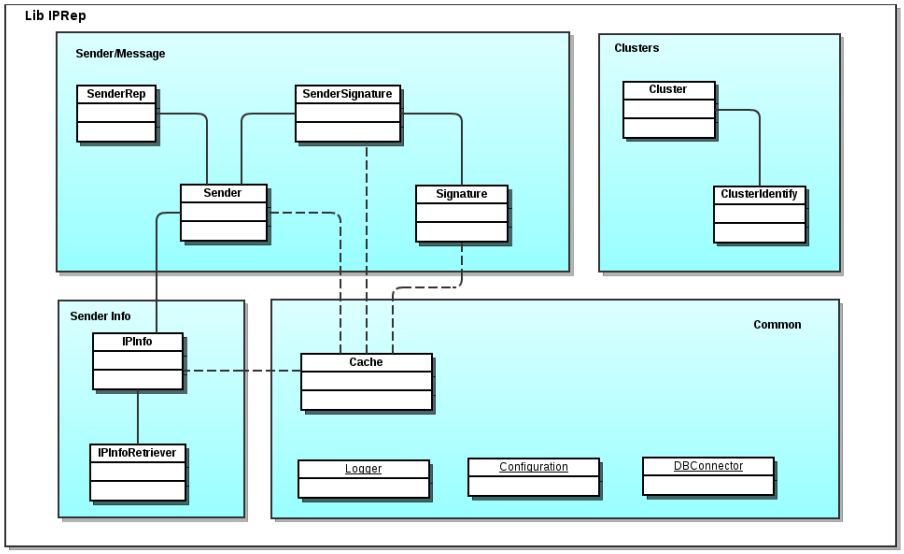

Figura 4.4: Arquitectura da biblioteca de reputação

entradas que estão dessincronizadas com a base de dados (*dirty*), e guarda-as na base de dados. A operação *expire* remove da *cache* entradas que já lá estejam há mais do que determinado tempo.

A operação de *load* é corrida sempre que chegam novas entradas ao *iprep-daemon*, mas certas operações como a *save* e a *expire*, têm de ser corridas periodicamente. Além destas, o *iprep-daemon* também é responsável por expirar assinaturas com mais de determinado tempo na base de dados, uma vez que o número de assinaturas cresce muito rapidamente. A partir da informação guardada na base de dados, o *ipinfo-daemon* percorre os IPs que tenham informação em falta na tabela IPInfo, para tentar obter essa informação, e o *ipcluster-daemon* tenta periodicamente descobrir *clusters* de *zombies*. No final deste fluxo, a informação guardada na base de dados é acedida através de pedidos DNS ao *Spike Aggregator* por parte de clientes que queiram saber a reputação de um IP ou se um IP pertence a algum *cluster* de *zombies*.

Este três *daemons* mencionados acima, fazem uso de uma biblioteca comum, que contém os métodos necessários para fazer o processamento das novas entradas e as actualizações na base de dados. A arquitectura desta biblioteca está representada na figura 4.4. Esta arquitectura foi construída tendo o modelo de dados em vista, de modo a que cada entidade seja tratada pela sua própria classe.

A *Cache* é uma classe genérica que se adapta a qualquer tipo de dados que necessite de fazer uso de uma *cache*. Isto permite que haja as quatro *caches* mencionadas anteriormente, servindo cada uma para guardar temporariamente os dados que serão posteriormente armazenados nas suas tabelas correspondentes na base de dados. No

momento em que chegam novas entradas ao *iprep-daemon*, estas são imediatamente processadas e os dados resultantes são colocados nas *caches*. Quando as actualizações periódicas são despoletadas, cada *cache* irá guardar nas suas tabelas correspondentes as entradas dessincronizadas. Para isto, as classes *Sender*, *Signature*, *SenderSignature* e *IPInfo*, apenas contém os métodos que lhes permitem interagir com a respectiva tabela na base de dados (*create*, *retrieve*, *update* e *delete*).

Para além destas classes, existem as que fazem o processamento dos dados. A classe *SenderRep* contem a heurística de reputação, sendo ela responsável por calcular os valores de reputação dos IPs a partir do histórico de envio de *emails* de cada um. A classe IPInfoRetriever contem os métodos que fazem a resolução de IPs para PTR *records* e que obtém o *asn*, prefixo de rede, registo, data de registo e código de país dos IPs. Existe ainda a classe *ClusterIdentify* que é responsável por descobrir que IPs fazem parte de *clusters* de *zombies* e por fazer a gestão dos *clusters*.

Capítulo 5

Resultados

Neste capítulo são apresentados os resultados obtidos pelas novas soluções face às actuais. A secção 5.1 compara a capacidade de processamento de *emails* por parte das duas soluções e apresenta a vantagem da adição do *milter* à solução. Na secção 5.2, é também feita uma comparação entre a performance das duas soluções, seguida de comparações relativas a valores de reputação, finalizando com uma comparação entre a capacidade de inserir entradas na base de dados utilizando duas técnicas diferentes.

5.1 *Spike Client*

O *Spike Client* foi totalmente redesenhado com o intuito de obter alguns ganhos em termos de flexibilidade e de performance. No capítulo anterior procurou-se evidenciar os ganhos de flexibilidade obtidos. Nesta secção apresentam-se os resultados de alguns testes que comparam a performance do sistema actual e do novo sistema.

5.1.1 Performance

No novo desenho do *spike client* há duas alterações que foram fundamentais na obtenção de aumento de performance: a alteração da linguagem de programação de Perl para C++, e a adição do *milter*. Tipicamente a adição de uma nova camada de abstracção não ajuda propriamente a aumentar a performance de um sistema, mas neste caso, o *milter* acaba por ser benéfico neste aspecto, pois este mantém uma *pool* de *threads* que chamam a *libspike* para cada *email* que chega ao MTA.

Para fazer a comparação de performance entre os dois sistemas, mediu-se a quantidade de *emails* que a actual solução consegue processar por segundo, e a quantidade de *emails* que a nova solução processa por segundo, com e sem o *milter*. O histograma da figura 5.1 mostra os ganhos obtidos em termos de performance.

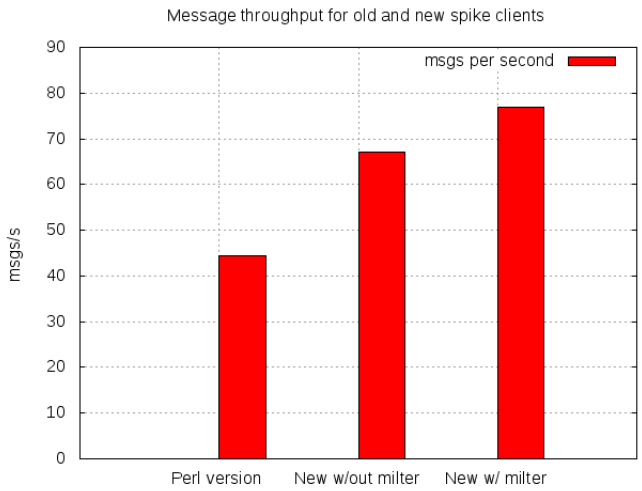

Figura 5.1: Capacidade do *Spike client* actual e novo em mensagens por segundo

Para realizar este teste, foi utilizado o *postal*, que é um programa que testa a capacidade de débito de um servidor de *mail*. Este programa envia *emails* de conteúdo aleatório e de tamanho configurável para o servidor de *email*, e a cada minuto que passa mostra a quantidade de mensagens que foi possível enviar durante esse minuto. No caso particular do teste da nova solução do *spike client* sem o *milter* não foi utilizado o *postal*, pois na nova solução é através do *milter* que a *libspike* fica integrada com o MTA. Assim o teste realizado para a nova *libspike* sem o *milter* consistiu em processar um conjunto de amostras em ciclo e medir o tempo que demora. Apesar do teste ter uma natureza diferente, permite dar uma ideia da vantagem da mudança de linguagem de programação.

Podemos ver pelos resultados, que a simples mudança de linguagem permitiu obter um aumento de performance considerável, e que a adição do *milter* subiu a capacidade de processamento da *libspike* quase para o dobro do que a solução actual é capaz. Num ambiente empresarial em que o fluxo de *mails* chega facilmente aos milhares por segundo estes resultados continuariam a ser insuficientes, mas é preciso ter em conta que quanto maior o fluxo de *mails*, mais MTAs serão utilizados pelas empresas. No entanto, o aumento de performance obtido com o novo *spike client* permite reduzir o número de MTAs, e com isto, reduzir o nível de energia despendido, ou então, mantendo os MTAs todos, permite suportar picos de carga no fluxo de *emails* que podem acontecer em certas alturas do ano.

5.2 *Spike Sender Reputation*

Tal como no *spike client*, os requisitos de flexibilidade e performance são também necessários para o novo *spike aggregator*. Nesta secção são apresentados os resultados dos testes que comparam a forma como os dois *aggregators* se comportam.

5.2.1 Contagens

Ao longo de 12 dias, foi realizado um teste em que os dois *aggregators* tiveram de processar as mesmas entradas que chegam ao sistema actualmente em produção. Este teste serviu não só para comparar a quantidade de informação armazenada por cada sistema (que é apresentada nesta secção) mas também para verificar como evoluem os valores de reputação com a nova heurística (na secção seguinte). Nesta secção é apresentada a quantidade de *ham* e *spam* processados por cada sistema, e a quantidade de *zombies* e *botnets* que cada sistema conseguiu detectar.

5.2.1.1 *Ham* e *Spam*

O gráfico da figura 5.2 ilustra a quantidade de entradas que representam *ham* e *spam* processadas por cada sistema.

Podemos ver que a quantidade de *ham* é muito superior à quantidade de *spam* (cerca de 5 vezes maior). Na realidade este rácio não reflecte a realidade dos *emails* que circulam na internet, pois as entradas apenas são classificadas como *spam*, se a assinatura contida na entrada constar da base de dados de *fingerprinting*. Muitas das entradas, embora sejam consideradas *ham*, são na realidade *spam*. As heurísticas de reputação (tanto a actual como a nova) conseguem mesmo assim atribuir má reputação à maioria dos emissores, uma vez que há outros factores que são tidos em conta.

Outro aspecto que podemos observar nos resultados deste teste, é o facto do novo sistema ter processado uma quantidade consideravelmente mais alta de entradas, do que o sistema actual. Em teoria ambos os sistemas deveriam ter o mesmo número de entradas processadas, uma vez que ambos receberam exactamente as mesmas entradas como *input*. No entanto é possível que haja alguma limitação no sistema actual, que faz com que o nível de carga a que o sistema está sujeito, já seja demasiada para a sua capacidade de processamento. Normalmente o número de entradas a ser processadas difere bastante em dois dias diferentes, principalmente se tivermos em conta fins-de-semana, em que o fluxo de *email* é bastante mais baixo. A evolução da quantidade de entradas processadas pelo sistema actual é demasiado linear, o que leva a crer que muitas das entradas estão a ser descartadas por falta de capacidade de processamento.

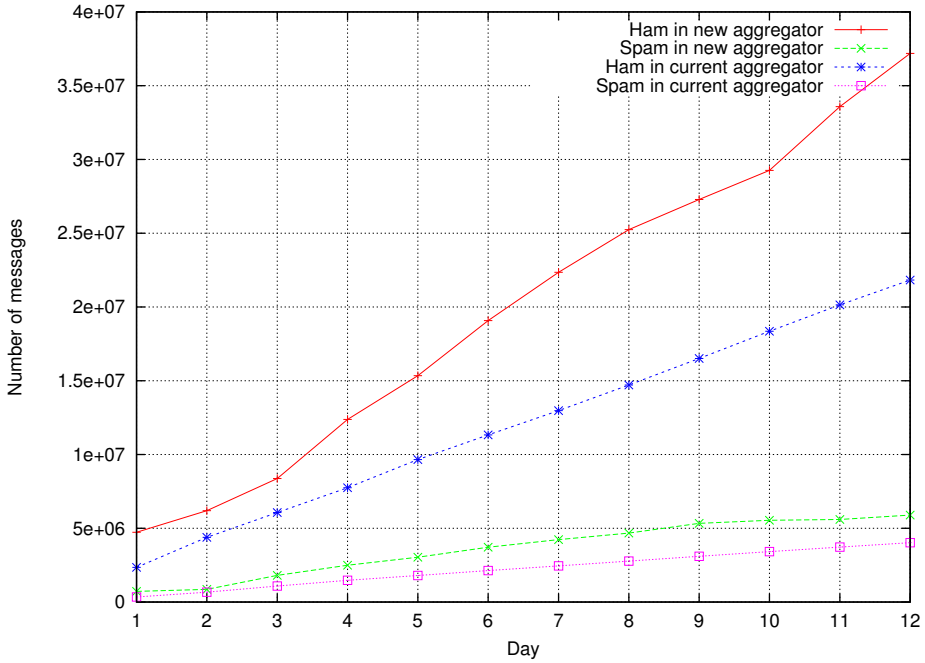

Figura 5.2: Quantidade de *ham* e *spam* ao longo de 12 dias nos dois sistemas

5.2.1.2 *Zombies* e *Botnets*

Tal como já foi mencionado anteriormente, um dos objectivos deste trabalho era alterar o algoritmo de detecção de *clusters* de *zombies*, de forma a tirar proveito do facto das assinaturas estarem divididas em quatro partes. O gráfico da figura 5.3 representa a quantidade de *zombies* detectados por cada sistema nos vários dias do teste.

Podemos ver no gráfico um início lento na descoberta de *clusters* por parte do novo algoritmo. Uma vez que o novo algoritmo procura IPs diferentes que tenham enviado assinaturas iguais ou parecidas, este acaba por ter um início mais demorado, pois o processamento que tem de ser feito, é consideravelmente mais pesado. Apesar disto, à medida que a base de dados vai tendo mais assinaturas, o novo algoritmo acaba por conseguir descobrir mais *zombies* do que o algoritmo antigo. Em ambos os algoritmos, as quedas nas contagens acontecem por expiração dos *clusters*. No entanto o sistema actual difere ligeiramente do novo sistema na forma como expira os *clusters*, pois estes são expirados quando estão inactivos durante mais do que determinado tempo, enquanto que no novo, o que é tido em conta, é o tempo de inactividade das assinaturas que representam os *clusters*.

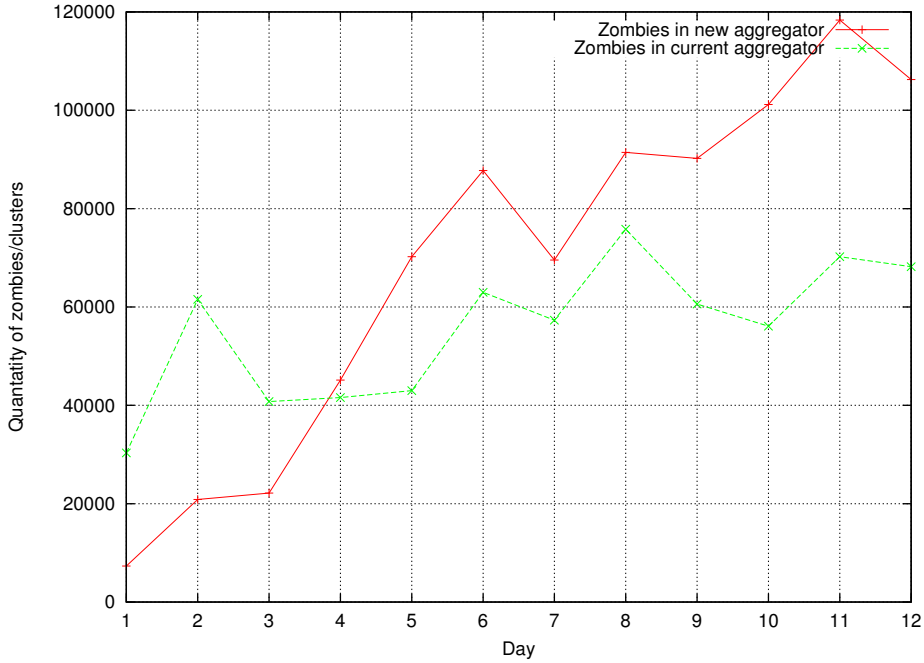

Figura 5.3: Quantidade de *zombies* descobertos pelos dois sistemas

5.2.2 Heurísticas de reputação

Como já foi mencionado anteriormente, outra grande alteração no sistema de reputação, foi na heurística de reputação. Com a utilização de médias móveis exponenciais para efectuar os cálculos, esperava-se conseguir que fosse dada mais importância ao histórico mais recente do comportamento dos emissores, e que a recuperação/perda de reputação fosse mais justa.

Nesta secção apresentam-se e analisam-se os resultados de um teste realizado simultaneamente pelos dois sistemas ao longo de doze dias com tráfego real. São também analisados os resultados de um teste que pretende verificar, através de entradas escolhidas propositadamente, como é que os emissores no novo sistema ganham e perdem reputação.

5.2.2.1 Recuperação e perda de reputação

O principal objectivo deste teste, é verificar como é que o valor de reputação de um IP evolui, à medida que este envia *emails* e à medida que o tempo passa. Para isto, testaram-se dois IPs, um associado a um PTR bom e outro associado a um PTR duvidoso, e realizaram-se as mesmas acções para os dois IPs. Estas acções podem ser envio de um certo número de *hams*, envio de um certo número de *spams* e actualização periódica de reputação. Esta última acção, implica passagem de tempo,

mais propriamente, 1 dia. Há vários factores que se procuram verificar com este teste. A forma como o valor de reputação evolui depende não só do número de *hams* e *spams* que um IP envia, mas também do seu nível de actividade e inactividade (e obviamente do PTR a que está associado).

O gráfico da figura 5.4 mostra os valores de reputação obtidos por efectuar as acções numeradas de 1 a 18 para um IP com bom PTR e para um IP com PTR duvidoso. As acções usadas para efectuar o teste foram as seguintes:

1. Envio de 5 *hams*
2. Envio de 5 *spams*
3. Actualização periódica
4. Envio de 5 *spams*
5. Actualização periódica
6. Envio de 5 *hams*
7. Actualização periódica
8. Actualização periódica
9. Actualização periódica
10. Envio de 10 *hams*
11. Envio de 5 *spams*
12. Actualização periódica
13. Envio de 5 *hams*
14. Envio de 10 *spams*
15. Actualização periódica
16. Actualização periódica
17. Actualização periódica
18. Actualização periódica

Facilmente podemos constatar que ambos os IPs têm uma evolução bastante semelhante, mas com níveis de reputação diferentes. Isto faz com que IPs associados a PTRs duvidosos tenham muito menos hipótese de ter reputação alta. Os valores de reputação com que cada IP começa, já têm em conta o envio de 5 *hams* inicial. Uma vez que no novo sistema a reputação é logo (re)calculada no momento em que o IP é

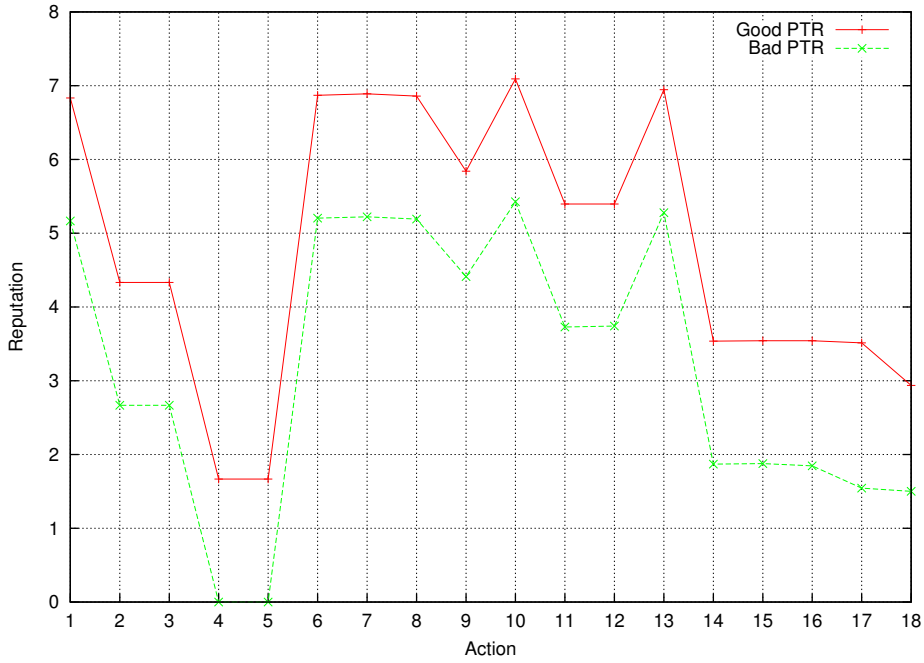

Figura 5.4: Evolução de dois emissores com PTR bom e PTR duvidoso

inserido ou actualizado na *cache* (em contraste com o sistema actual, que tem que esperar pelas actualizações de reputação periódicas), o envio de *emails* traduz-se numa subida ou descida imediata do valor de reputação.

A importância dada ao histórico mais recente pode ser facilmente observada na acção 6, em que após início de um novo período, a reputação é recuperada rapidamente, em contraste com as acções 10 e 11, em que o emissor após enviar 10 *hams* seguidos de 5 *spams* vê a sua reputação pouco prejudicada, pois o envio de *ham* recente é tido em conta.

Nas últimas acções, podemos ainda verificar o efeito que a inactividade tem sobre a reputação dos IPs, fazendo com que esta comece a descer gradualmente.

5.2.2.2 Distribuição dos valores calculados

Este teste, realizado ao longo de doze dias, põe à prova a disponibilidade e a tolerância a picos de carga dos dois sistemas, mas acima de tudo, pretende demonstrar as diferenças entre as heurísticas dos dois sistemas, apresentado a forma como os vários emissores que vão sendo guardados nas bases de dados, ficam associados a diferentes valores de reputação. As heurísticas têm três diferenças fulcrais, que se traduzem nos resultados observados: (1) a importância dada ao histórico mais recente de envio dos emissores; (2) a forma como emissores inactivos muito tempo são expirados; (3) a

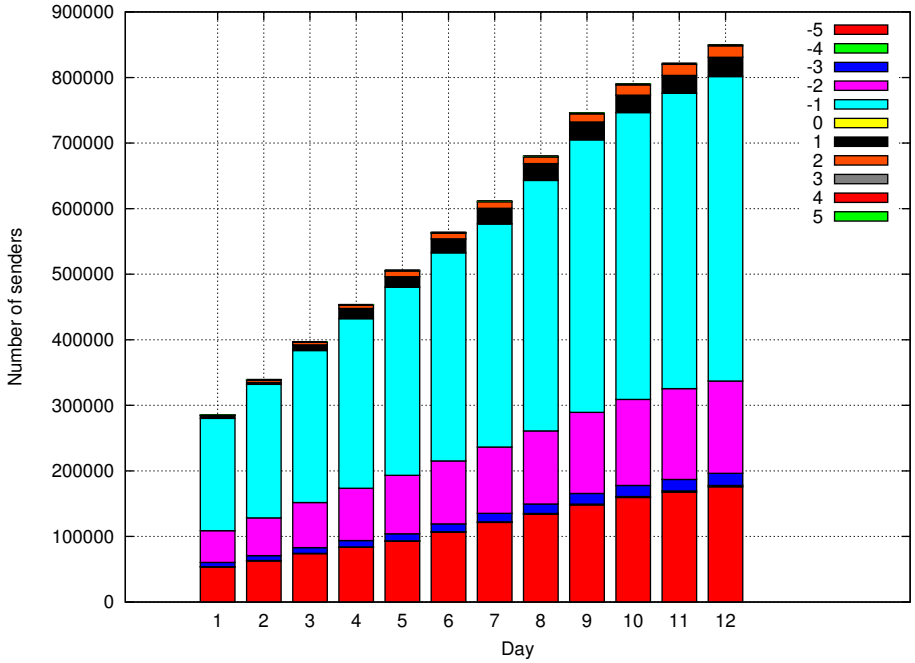

Figura 5.5: Distribuição dos valores de reputação do sistema actual ao longo de 12 dias

escala utilizada. A primeira diferença, refere-se ao facto da nova heurística dar mais importância ao histórico de envio mais recente dos emissores, em contraste com a heurística actual em que um emissor para recuperar de uma má reputação, tem que enviar tantos *hams* como *spams* já enviados, e vice-versa. A segunda diferença diz respeito ao facto da nova heurística remover IPs que ficam com as médias de envio de *ham* e *spam* (simultâneamente) abaixo de um certo limiar (neste caso 0,1 de média), enquanto que no sistema actual, a expiração de emissores é feita por um script externo que remove IPs inactivos. A última diferença refere-se à utilização de uma escala contínua no sistema novo (0 a 10) *versus* uma escala discreta no sistema actual (-5 a 5). Os gráficos das figuras 5.5 e 5.6 representam as distribuições dos valores de reputação em cada um dos dias do teste, no sistema actual e no sistema novo respectivamente.

Podemos ver no gráfico do sistema actual (figura 5.5) que há muito poucos emissores com reputação positiva. Tendo em conta os resultados apresentados na secção 5.2.2.1, onde vemos que a quantidade de *ham* que chega aos sistemas é muito superior à quantidade de *spam*, estes resultados podem parecer estranhos, mas isto acontece porque a maior parte dos emissores estão associados a PTRs duvidosos, e este factor é suficiente para prejudicar emissores com poucos emails enviados, mesmo que esses emails estejam classificados como *ham*. Além disto, quando a reputação de um IP é zero, a heurística assume automaticamente uma reputação de -1.

Outro facto facilmente observável no gráfico da figura 5.5, é a forma bastante linear como o sistema evolui. Isto deve-se principalmente ao facto de não ser dada mais importância ao histórico mais recente dos emissores, fazendo com que os IPs mais activos, na maioria das vezes tenham mais dificuldade em recuperar de má reputação obtida devido a emissores que enviaram muito *spam* a partir desses IPs. O facto da escala ser discreta, provavelmente também faz com que seja difícil um IP, com bastante histórico de envio, deixar de estar associado a um certo valor de reputação.

De notar que a implementação de um sistema que utilize o DNS de reputação da forma que é aconselhada pela *AnubisNetworks[1]* ignora reputação com valor 0 (valor ao qual nem há IPs associados), -1 e 1. Isto significa que para IPs incluídos nas maiores fatias do gráfico da figura 5.5, não há informação sobre o que se deve fazer a *emails* provenientes deles. Como foi mencionado anteriormente, grande parte destes emissores têm um histórico de envio baixo, pelo que é difícil ter a certeza se os seus *emails* devem ou não ser rejeitados.

No gráfico da figura 5.6 notamos que a nova heurística demora algum tempo a adaptar-se, uma vez que no início há pouca informação sobre os emissores (PTRs desconhecidos inicialmente e poucos *emails* enviados pela grande maioria dos IPs). A nova heurística é muito benevolente e muito severa para IPs com pouco histórico de envio, pelo que inicialmente grande parte dos IPs estão associados a reputações entre 1 e 2, e entre 6 e 7. A partir do terceiro dia, este problema começa a desaparecer. Este teste pretendia mostrar exactamente como é que a nova heurística atribui os valores de reputação aos IPs, mas tendo em conta que não se pode decidir nada a cerca de IPs que têm um histórico de envio baixo, uma possível acção a tomar em relação estes, seria atribuir-lhes uma reputação de 5 para que possam ser ignorados quando se questiona a sua reputação.

Na nova heurística os valores de reputação estão mais equilibrados entre reputação positiva e negativa. Isto não significa que com o novo sistema conseguimos rejeitar menos *emails*. Tendo em conta que com a heurística actual os IPs com reputação -1 são ignorados, se olharmos por exemplo para o último dia do teste, só podemos verdadeiramente rejeitar *emails* provenientes de cerca de trezentos mil (300000) IPs. Também com a nova heurística conseguiríamos rejeitar *emails* provenientes do mesmo número de IPs, tendo o último dia do teste como referência.

Em contraste com a heurística usada actualmente, há mais diversidade nos valores de reputação, uma vez que os emissores perdem ou ganham reputação mais rapidamente devido à importância dada ao histórico mais recente. As contagens de cada período são reiniciadas diariamente, fazendo com que, por exemplo, um IP que tenha enviado bastante *spam* num dia, terá hipótese de se redimir rapidamente no dia seguinte. A remissão durante o mesmo período também é obviamente possível como vimos no teste anterior, mas é mais complicada.

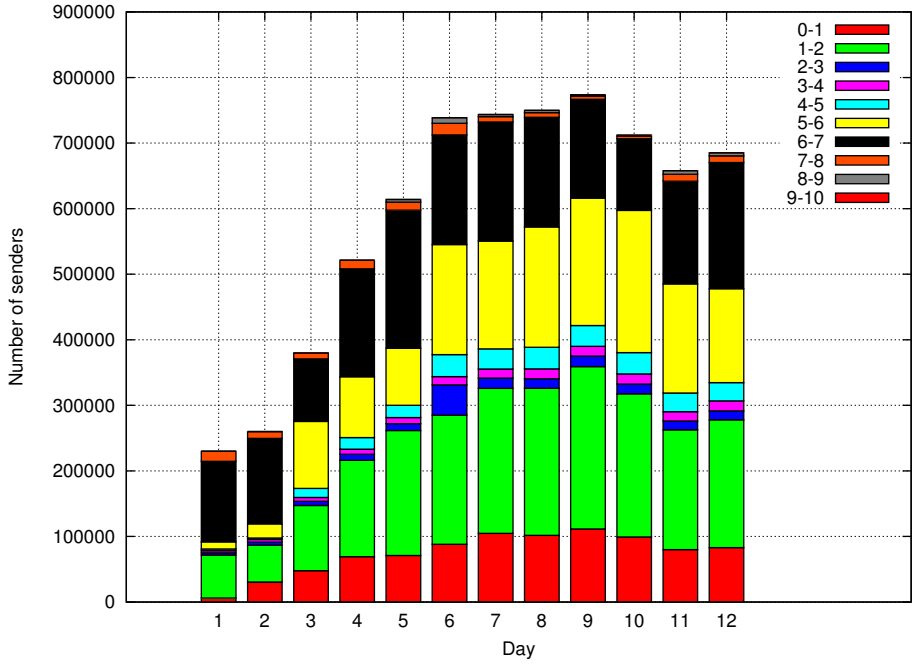

Figura 5.6: Distribuição dos valores de reputação do novo sistema ao longo de 12 dias

5.2.3 Inserções normais VS Inserções múltiplas

(o problema da base de dados como bottleneck: comparação entre utilização de inserções normais e inserções múltiplas)

Neste sistema, em que é utilizado uma base de dados onde estão constantemente a ser inseridas novas entradas, um dos maiores desafios é conseguir que as inserções sejam rápidas. Por muito rápido que se consiga carregar os dados para memória, o *bottleneck* acaba por ser sempre a escrita na base de dados. Se não for possível guardar na base de dados mais entradas do que as que vão chegando, a memória começa a crescer indefinidamente. Ao longo do desenvolvimento do novo sistema, foram experimentados vários tipos de inserções que são suportadas pelo MySQL. O gráfico da figura 5.7 ilustra os tempos demorados por cada um dos tipos de inserções, para 50000 inserções. Estes teste foi feito para uma base de dados vazia e para uma base de dados com centenas de milhares de dados em cada tabela, uma vez que, como se pode ver no gráfico, os tempos demorados são bastante diferentes.

Inicialmente utilizaram-se inserções simples, nas quais é feito um pedido à base de dados por cada entrada que se quer inserir ou actualizar. Este método revelou-se bastante ineficiente, pois a quantidade de pedidos que têm de ser feitos à base de dados é demasiado grande. Para responder a este problema, procurou-se outra forma de fazer as inserções mais rapidamente, e implementou-se um esquema de inserções

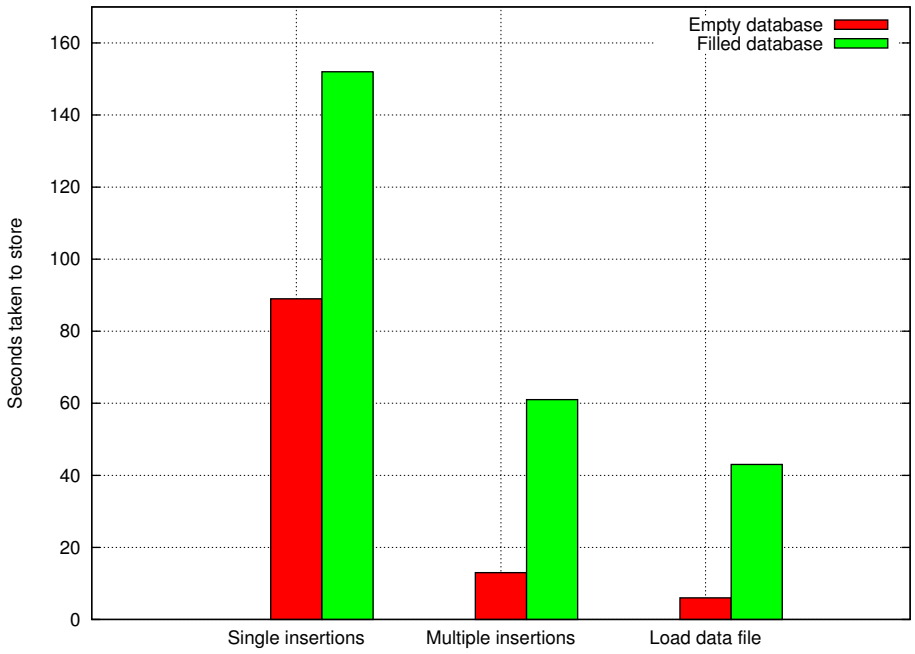

Figura 5.7: Tempo demorado por cada tipo de inserção a inserir 50000 entradas

múltiplas, em que na mesma *query* SQL, são enviados todos os tuplos a ser inseridos ou actualizados. Numa base de dados vazia este método parecia ser bastante eficaz, mas o seu comportamento numa base de dados cheia (principalmente com tráfego real) continuava não ser suficiente. Finalmente, implementou-se um método de inserção, que consiste em carregar para a base de dados, um ficheiro com todas as entradas a ser inseridas e actualizadas. Tal como nos dois casos anteriores, a sua performance também se degrada um pouco quando a base de dados está mais cheia, mas provou ser suficiente para processar os dados de tráfego real e para aguentar alguns picos de carga.

De notar que este teste foi feito com entradas aleatórias geradas por um script, que estão longe de reflectir entradas de tráfego real, e como tal, o comportamento do sistema acaba por não ser o normal. No entanto esta era a única forma de controlar a quantidade de entradas processadas por cada método de inserção. Num cenário real, o método de inserções múltiplas acaba por ser mais lento do que método inserções simples, sendo esta a principal razão que nos levou a decidir descartar este método. Em contraste com isto, o método de inserção por carregamento de ficheiros provou ser mais do que suficiente num cenário real, sendo portanto esta a escolha final.

Capítulo 6

Conclusões

O primeiro objectivo deste trabalho, foi a refactorização da arquitectura do *spike client*, mantendo o protocolo de comunicação com o *spike node* e a forma como as assinaturas são geradas intactos, mas ao mesmo tempo, procurando obter mais flexibilidade e mais performance na nova solução. No sentido de concretizar esta meta, optou-se por mudar para uma linguagem de programação (C++) que facilitasse a modularização da solução, sem comprometer a performance. Para obter o ganho de performance necessário, acrescentou-se uma camada de abstracção (*milter*), que veio paralelizar o processamento das mensagens, conseguindo assim solucionar a integração da *libspike* com o MTA e obter o ganho de performance desejado.

Tal como o *spike client*, também o *spike aggregator* exigia uma nova arquitectura mais flexível e com melhor desempenho que o actual. Para tal foram criados novos modelos de dados e de reputação tendo em vista estas necessidades. Havendo mais liberdade para fazer alterações do que no *spike client*, criaram-se também novos algoritmos para calcular os valores de reputação e para efectuar a descoberta de *botnets*. Nos resultados obtidos, podemos observar os benefícios que o novo desenho veio trazer. O ganho em performance é visível nas contagens de *ham* e *spam* efectuadas pelos dois sistemas, onde se pode verificar que o sistema actual já não tem capacidade suficiente para processar todas as entradas que chegam. Também a nova heurística trouxe algumas vantagens, nomeadamente a forma como é dada importância ao histórico de envio mais recente dos emissores, embora ainda haja espaço para melhoria neste aspecto. A detecção de *botnets* passou a utilizar, nesta nova solução, um algoritmo, que apesar de mais pesado em termos de processamento, tem a capacidade de detectar mais *botnets* e *zombies* do que o algoritmo usado actualmente. Este peso acrescido era de esperar, uma vez que a alteração feita implica o agrupamento de assinaturas parciais, mas mesmo assim isto revelou-se compensador.

Os objectivos deste projecto passavam por melhorar uma solução que por si só já

era bastante eficaz a resolver os problemas que motivaram a sua construção, mas que apesar disto precisava de uma revisão para se adaptar aos novos desafios proporcionados por estes problemas. Consideramos que de um modo geral, conseguimos atingir os objectivos propostos para este trabalho.

6.1 Trabalho Futuro

Os resultados da nova heurística de reputação mostram que esta tem alguns problemas a calcular valores de reputação justos para emissores com um histórico de envio baixo. Isto acaba por ser normal, uma vez que é difícil decidir alguma coisa sobre um emissor, se houver pouca informação acerca do seu envio de *emails*. Como foi mencionado anteriormente, uma hipótese para resolver este problema seria atribuir um valor de reputação igual a 5 a emissores com baixo histórico de envio, para que estes possam ser ignorados até terem enviado *emails* suficientes para se saber efectivamente qual a sua reputação. Num futuro imediato, esta é uma hipótese que deverá certamente ser testada.

Está neste momento a ser desenvolvida uma plataforma que irá integrar todos os sistemas do *Spike*. O próximo passo para o *Spike client* e *aggregator* desenvolvidos neste trabalho, assim como para as outras componentes que fazem parte do *Spike*, será fazer a integração nesta nova plataforma.

Bibliografia

[1] Anubis mailspike implementation - available dns zones web page. http://labs.anubisnetworks.com/anubis/implementation_zones.html.

[2] The definition of spam web page. http://www.spamhaus.org/definition.html.

[3] Distributed checksum clearinghouses reputations web page. http://www.rhyolite.com/dcc/reputations.html.

[4] Distributed checksum clearinghouses web page. http://www.rhyolite.com/dcc/.

[5] Milter web page. https://www.milter.org/.

[6] Pyzor - open source collaborative spam filtering web page. http://sourceforge.net/apps/trac/pyzor/wiki/TitleIndex.

[7] Spam filter reviews, anti spam tips, advice and spam filter ratings web page. http://www.whichspamfilter.com/TypesOfFilters.htm.

[8] Vipul's razor collaborative spam filtering network web page. http://razor.sourceforge.net/docs/.

[9] Ten sneaky things a spammer will do. http://www.sonicwall.com/us/products/resources/10661_13586.htm 2009.

[10] D. Alperovitch, P. Judge, and S. Krasser. Taxonomy of email reputation systems. pages 27 –27, jun. 2007.

[11] Ion Androutsopoulos, John Koutsias, Konstantinos V. Chandrinos, Konstantinos V. Ch, George Paliouras, and Constantine D. Spyropoulos. An evaluation of naive bayesian anti-spam filtering. pages 9–17, 2000.

[12] Daniel Lowd and Christopher Meek. Good word attacks on statistical spam filters. In *In Proceedings of the Second Conference on Email and Anti-Spam (CEAS)*, 2005.

[13] Paul Resnick, Richard Zeckhauser, Eric Friedman, and Ko Kuwabara. Reputation systems. *Comm. ACM*, 43:45–48, 2000.

[14] Bradley Taylor. Sender reputation in a large webmail service, 2007.

[15] Meng Weng Wong. Sender authentication - what to do. maawg whitepaper, 2004.