# BeShort: Um algoritmo para encurtamento de URLs

Pedro Paulo Simões Freitas

Orientador: Fabrício Benevenuto
Universidade Federal de Ouro Preto

Dissertação submetida ao
Instituto de Ciências Exatas e Biológicas da
Universidade Federal de Ouro Preto
para obtenção do título de Mestre em Ciência da Computação

B866b Freitas, Pedro Paulo Simões.
BeShort [manuscrito] : um algoritmo para encurtamento de URLs / Pedro Paulo Simões Freitas – 2012.
xx, 53 f.: il. color.; grafs.; tabs.

Orientador: Prof. Dr. Fabrício Benevenuto.

Dissertação (Mestrado) - Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Biológicas. Departamento de Computação. Programa de Pós-graduação em Ciência da Computação.
Área de concentração: Sistemas de Computação

1. Redes de computadores - Teses. 2. Spam (Mensagens eletrônicas) - Teses. 3. Redes sociais - Teses. I. Universidade Federal de Ouro Preto. II. Título.

CDU: 004.7:004.057.4

Catalogação: sisbin@sisbin.ufop.br

## Ata da Defesa Pública de Dissertação de Mestrado

Aos 14 dias do mês de dezembro de 2012, às 10 horas na Sala de Seminários do Departamento de Computação do Instituto de Ciências Exatas e Biológicas (ICEB), reuniram-se os membros da banca examinadora composta pelos professores: **Prof. Dr. Fabrício Benevenuto de Souza (presidente e orientador), Prof. Dr. Ricardo Augusto Rabelo Oliveira, Prof. Dr. Joubert de Castro Lima e Profa. Dra. Jonice de Oliveira Sampaio**, aprovada pelo Colegiado do Programa de Pós-Graduação em Ciência da Computação, a fim de arguirem o mestrando **Pedro Paulo Simões Freitas**, com o título **"BeShort: Um algoritmo para encurtamento de URLs"**. Aberta a sessão pelo presidente, coube ao candidato, na forma regimental, expor o tema de sua dissertação, dentro do tempo regulamentar, sendo em seguida questionado pelos membros da banca examinadora, tendo dado as explicações que foram necessárias.

**Recomendações da Banca:**

**(X) Aprovada sem recomendações**

**( ) Reprovada**

**( ) Aprovada com recomendações:** _____

**Banca Examinadora:**

Fabrício Benevenuto de Souza
Prof. Dr. Fabrício Benevenuto de Souza

Prof. Dr. Ricardo Augusto Rabelo Oliveira

Prof. Dr. Joubert de Castro Lima

Jonice Oliveira
Profa. Dra. Jonice de Oliveira Sampaio

Haroldo G. Santos
Prof. Dr. Haroldo Gambini Santos
Coordenador do Programa de Pós-Graduação em Ciência da Computação
DECOM/ICEB/UFOP

**Ouro Preto, 14 de dezembro de 2012.**

*Dedico este trabalho a meus pais,* Pedro *e* Pilar*,*
*pelo incentivo e amor incondicional.*

# BeShort: Um algoritmo para encurtamento de URLs

## Resumo

Microblogs como o Twitter são sistemas sociais voltados unicamente para a postagem de mensagens com no máximo 140 caracteres. Com o grande uso de mensagens curtas na Web o uso de encurtadores de URLs está se tornando cada vez mais comum. Sistemas encurtadores traduzem uma URL com dezenas de caracteres em uma nova URL, tipicamente com poucos caracteres e redirecionam requisições da URL encurtada para a URL longa original. Apesar de extremamente eficiente, esses serviços podem introduzir atrasos para seus usuários e têm sido amplamente utilizada para ofuscar spam, phishing e malware. Esse trabalho apresenta o BeShort, um algoritmo para encurtamento de URLs capaz de evitar tais problemas. Nossa abordagem consiste em substituir partes frequentes ocorridos (ex. "www" e "http:") por caracteres UTF-8, normalmente não utilizados em URLs. Para testar nossa abordagem, utilizamos uma base contendo 50 milhões de URLs de dois serviços encurtadores de URL bastante populares. Nossos resultados mostram que o BeShort consegue taxas de encurtamento tão eficientes quanto as taxas praticadas pelas arquiteturas atuais.

# BeShort: An algorithm for shortening URLs

## Abstract

Microblogs like Twitter are social systems designed to allow users to post messages containing no more than 140 characters. With the wide use of short messages on the Web, the useof URL shorteners are increasingly becoming popular. These systems translate a shortened URL into a new URL, typically with few characters, and redirect requests that target the shortened version of the URL to the original long URL. Although extremely efficient, the centralized architecture of such services can introduce delays to users and have been widely used as a way to obfuscate spam, phishing and malware. This paper presents BeShort, a distributed approach for shortening URLs able to avoid such problems. Our approach consists of replacing frequently terms (e.g. "www" e "http:") for UTF-8 characters that are usually not used in URLs. To test BeShort we built a dataset containing 50 million URLs of two popular URL shortening services. Our results show that the BeShort obtains compression rates as efficient as the rates obtained by existent approaches.

# Agradecimentos

Primeiro quero agradecer a *Deus* por me propiciar a realização de mais um sonho.

Agradeço a *meus pais*, pelo incentivo.

Agradeço a *meus irmãos*, pelo apoio.

Agradeço a *minha namorada* Aline, pelo carinho e companheirismo.

Agradeço a *meus primos*, em especial Maninho e Victor Hugo.

Agradeço a *meus amigos*, em especial Paulo Henrique e Guilherme.

Agradeço a *meu orientador* Fabrício, pelo aprendizado.

Agradeço aos *professores do DECOM/UFOP*.

Agradeço a *todos companheiros* da República Maternidade.

Agradeço a *todos familiares*.

Muito Obrigado a todos.

# Sumário

# Lista de Figuras

# Lista de Tabelas

# Lista de Algoritmos

*"Penso noventa e nove vezes e nada descubro; deixo de pensar, mergulho em profundo silêncio - e eis que a verdade se me revela."*

— Albert Einstein

# Lista de Siglas, Acrônimos e Abreviaturas

| | |
|---|---|
| WWW | *World Wide Web* |
| URL | *Uniform Resource Locator* |
| HTTP | *HyperText Transfer Protocol* |
| CDNs | *Content Distribution Networks* |
| UTF | *Unicode Transformation Formats* |
| UTF-8 | *Unicode Transformation Formats* - 8 *bit* |
| UTF-16 | *Unicode Transformation Formats* - 16 *bit* |
| UTF-32 | *Unicode Transformation Formats* - 32 *bit* |
| ASCII | *American Standard Code for Information Interchange* |
| $Freq \times Tam$ | Frequência multiplicado por Tamanho |
| $Freq - Sub$ | Frequência subtraído Subpalavra |
| UTF-Total | Dicionário com todos caracteres possíveis do padrão Unicode |
| UTF-Parcial | Dicionário com apenas os caracteres utilizados nos principais idiomas do mundo |
| CDF | *Cumulative Distribution Function* |
| API | *Application Programming Interface* |
| JSP | *JavaServer Pages* |

# Capítulo 1

# Introdução

Desde seu início a Internet tem sido palco de uma série de novas aplicações incluindo a WWW e email. Atualmente, a Web tem recebido uma nova onda de aplicações associadas ao crescimento e proliferação das redes sociais online. Surgiram vários desses sistemas incluindo redes de profissionais (ex. LinkedIn[1]), redes de amizade (ex. Facebook[2] e Google+[3]), e redes voltadas para o compartilhamento de algum tipo específico de conteúdo como mensagens curtas (ex. Twitter[4]), diários e blogs (ex. LiveJournal[5]), fotos (ex. Flickr[6]) e vídeos (ex. YouTube[7]).

Redes sociais online têm atraído milhões de usuários. De acordo com a Nielsen Online [30], a mídia social já superou a troca de emails como a atividade online mais popular. Mais de dois terços da população online global visita ou participa de redes sociais e blogs. Como comparação, se o Facebook fosse um país, seus mais de 1 bilhão de usuários registrados colocariam esta aplicação como o terceiro país mais populoso do mundo [14]. No Twitter, um sistema que permite unicamente a postagem de mensagens curtas com no máximo 140 caracteres (tweets), recebe cerca de 500 milhões de mensagens por dia, que são enviadas a milhares de usuários [13].

O acesso a redes sociais através de dispositivos móveis, como *smartphones* (celulares) e *tablets*, é um recurso que vem sendo bastante utilizado pelos usuários. Um estudo feito

[1] http://br.linkedin.com/
[2] http://www.facebook.com
[3] https://plus.google.com/
[4] https://twitter.com/
[5] http://www.livejournal.com/
[6] http://www.flickr.com
[7] http://www.youtube.com

por [10], mostra que no Facebook e no Twitter os usuários passam mais tempo usando essas redes em dispositivos móveis do que em computadores tradicionais ou notebooks. Os usuários do Facebook, no mês de março, ficaram mais de sete horas olhando o site via celular, e em torno de seis horas pelo computador. No Twitter, os usuários móveis passaram mais de duas horas online, já em computadores o tempo de uso caiu para $20,4$ minutos. Com esse grande uso de *smartphones*, gera uma necessidade do emprego de mensagens cada vez mais curtas e postadas em tempo real.

O uso de mensagens curtas tem sido amplamente explorado nas redes sociais, permitindo que os usuários discutam sobre tudo, incluindo notícias, casualidades, suas opiniões, repercussão de eventos ou produtos [7]. Um exemplo é o Twitter que é utilizado pelos usuários para fazer campanhas políticas [27], promoção de negócios, onde as lojas criam contas para fazer divulgação de ofertas e informações de seus produtos, e na comunicação de eventos de emergência [21, 32, 34]. Nesse mesmo contexto, milhões de URLs são compartilhadas todos os dias [29], mudando a forma como as pessoas descobrem conteúdo na Web [31]. Várias redes sociais impõem um limite superior no tamanho da mensagem (ex. no Twitter a mensagem é limitada a 140 caracteres), levando os usuários a utilizar um serviço encurtador de URLs para economizar espaço de suas mensagens.

De fato, encurtar URLs vem se tornando uma das principais maneiras para a fácil disseminação e compartilhamento de URLs. Serviços encurtadores de URLs, como Bit.ly[8] e TinyURL,[9] estão se tornando cada vez mais comuns. Os encurtadores de URLs traduzem uma URL (que pode consistir de centenas de caracteres) em uma nova URL, tipicamente com poucos caracteres que retorna os códigos HTTP 301 ou 302 de redirecionamento para à URL longa original [1]. Por exemplo, o link http://nyti.ms/1VKbrC, ao ser acionado, irá redirecionar para o sítio Web original, que comparado com a versão encurtada, consiste em uma URL com mais do que o dobro de caracteres. Apesar do primeiro sistema, com popularidade notável, ter surgido em 2002, hoje usuários podem escolher uma imensa variedade de serviços [28].

Embora úteis, às URLs encurtadas introduzem alguns novos problemas e é nesse contexto que essa dissertação está inserida. A seguir, na Seção 1.1 discutimos esses problemas e apresenta nossos objetivos. A Seção 1.2 sumariza as principais contribuições dessa dissertação e a Seção 1.3 discute a organização dos demais capítulos.

[8] http://bit.ly
[9] http://tinyurl.com

## 1.1 Problemas e Objetivos

Nesta seção são descritos os principais problemas causados por URLs encurtadas, os quais, motivam essa dissertação. Em seguida, nossos objetivos são apresentados.

**Facilidade para *phishing* e *spam***: Serviços encurtadores de URL vêm sendo comumente utilizados como forma de esconder *spam* e *phishing*, o que vem sendo reportado em um grande número de trabalhos científicos recentes [4, 8, 19, 22, 24, 36]. Em particular, *phishers* utilizam URLs encurtadas para ofuscar suas URLs, conforme mostrado recentemente em [8]. Como exemplo, a figura 1.1 mostra um *tweet* real contendo uma URL que aparece em listas negras de *phishing*, ofuscada por uma URL do Bit.ly. Clicando na URL do *tweet*, usuários são levados a uma página que parece a página de *login* do Twitter. Tentados pela oferta de acessar o perfil de outros usuários, um usuário pode entrar com suas credenciais do Twitter e perder sua conta para *phishers*. De posse de uma conta real, *phishers* podem realizar ataques mais elaborados (ex. ataques para obter cartões de crédito ou contas de bancos) aos seguidores dos usuários com a conta comprometida [8].

**Atrasos de redirecionamento**: Os serviços encurtadores de URL mais utilizados atualmente, como o Bit.ly e TinyURL, encontram-se hospedados no exterior, exigindo, muitas vezes, redirecionamento para servidores localizados em regiões muito distantes. Tal redirecionamento pode impor severos atrasos no tempo de resposta, estes atrasos são quantificados na Seção 5.6.1. Além disso, por se tratarem de serviços gratuitos, a grande maioria desses encurtadores de URL não oferecem garantias sobre a qualidade de seus serviços e podem até mesmo não atender requisições, caso estejam sobrecarregados.


**Figura 1.1:** Um *tweet* com uma URL encurtada (utilizando bit.ly).

Esse cenário sugere a necessidade do desenvolvimento de uma arquitetura que seja capaz de encurtar URLs de forma a evitar os problemas apontados acima.

**Objetivo da dissertação:** Este trabalho visa investigar uma abordagem para encurtar URLs que dispensa o uso de redirecionamento para um servidor central. A ideia é que um algoritmo de encurtamento de URL seja executado no momento do envio da mensagem à rede social (o algoritmo encurta a URL) que, ao ser recebida, é expandida e exibida aos usuários em sua versão longa original.

## 1.2 Contribuições do Trabalho

A seguir serão apresentadas as principais contribuições dessa dissertação:

- Para a realização dos testes em nossa abordagem descentralizada, foram extraídas uma ampla coleção de URLs encurtadas de uma base completa do Twitter, contendo todos os *tweets* postados desde a criação do Twitter em 2006 até julho de 2009. A partir dessas URLs curtas foram encontradas suas respectivas URLs longas, com isso identificamos **67.324.019** pares de URLs curtas e suas versões longas. Essa base de dados é apropriada para o nosso propósito por se tratar de uma coleta completa do Twitter e não de uma amostra potencialmente tendenciosa. Pretendemos disponibilizar essa coleção de pares de URLs, pois pode ser de suma importância na contribuição para outros trabalhos.

- Avaliação de viabilidade de um novo método para encurtar URLs onde sua arquitetura apresenta-se de forma descentralizada, evitando os problemas dos encurtadores atuais. O método proposto obteve desempenho semelhante aos métodos praticados atualmente, demonstrando que é viável a realização de compressão com uma abordagem descentralizada.

- Construção de um protótipo do BeShort, que se conecta ao Twitter pela sua API, para a realização de teste reais e disponibilização do seu código. Acreditamos que a disponibilidade do código juntamente com a base de pares URLs, pode permitir não só a reprodução dos resultados desse trabalho, mas também a comparação da abordagem implementada do BeShort com abordagens futuras.

## 1.3 Organização dos Capítulos

O restante da dissertação está organizada da seguinte forma. O **Capítulo 2** aborda trabalhos relacionados, mostrando estudos que apontam a utilização das URLs curtas e confirmam seu uso para ofuscar URLs maliciosas. Em seguida, o **Capítulo 3** mostra como foi obtida a base de dados utilizada. Depois, no **Capítulo 4** apresentamos a arquitetura do BeShort comparando-a com a arquitetura praticada pelos serviços utilizados atualmente, mostrando também os passos para a criação de um dicionário, que é usado na compressão e descompressão das URLs. Posteriormente no **Capítulo 5** exibimos experimentos realizados com o BeShort, comparando-o aos serviços Bit.ly e TinyURL. No **Capítulo 6** é apresentado um protótipo do BeShort, seu funcionamento e apresenta algumas imagens que ilustram sua execução. Finalmente, no **Capítulo 7** concluímos o trabalho e abordamos trabalhos que poderão ser realizados futuramente.

# Capítulo 2

# Trabalhos Relacionados

Pelo fato de nenhuma proposta relacionada ao BeShort ter sido mostrada na literatura até o presente momento, neste capítulo, apresentaremos trabalhos que mostram como são utilizadas as URLs encurtadas, em quais aplicação estão sendo usadas e trabalhos que identificam e confirmam o uso de URLs curtas para uma melhor propagação de URLs que contém algum tipo de ataque malicioso (*spam*, *phishing*).

Antes de falar sobre trabalhos relacionados a encurtadores de URLs, falaremos sobre um trabalho que aborda a propagação de URLs em redes sociais. Rodrigues e colaboradores [31] proveem uma série de análises sobre os padrões de propagação de URLs entre os usuários do Twitter. Também quantificam o aumento de audiência que o uso de *retweets* pode causar a uma URL, eles mostram que domínios pouco populares na web podem se tornar populares através do espalhamento boca-a-boca no Twitter. O termo boca-a-boca refere-se ao espalhamento de informação pelas conversas entre os usuários das redes sociais. O trabalho ainda identifica características típicas da estrutura das árvores de propagação de informação nesse sistema e mostra que, no Twitter as árvores são mais largas do que profundas.

O trabalho mencionado ressalta o grande interesse por espalhamento de informações em sistemas como o Twitter e quantificam o volume de URLs compartilhadas nesses sistemas, sendo que a maioria dessas é postada de forma encurtada. De fato, Antoniades e colaboradores [1] mostram que cerca de 87% das URLs postadas no Twitter são encurtadas e que URLs encurtadas não são utilizadas apenas no Twitter, mas também em outros serviços como emails e outras redes sociais online. Além disso, os autores mostram que URLs encurtadas possuem vida curta em termos de popularidade, ou seja,

URLs possuem popularidade por intervalos pequenos de tempo e depois deixam de ser usadas. Tal observação sugere que sistemas encurtadores centralizados mantenham listas enormes de URLs que só possuem acessos em um curto intervalo de tempo. Os autores também concluem que os serviços encurtadores são bastante eficazes na redução de tamanho da URL. Outro ponto investigado é o atraso imposto por encurtadores de URL, que se mostrou significativo, chegando a ser 2 vezes superior do que o acesso direto à URL em algumas situações.

Um aspecto relacionado à grande popularidade do uso de encurtadores de URLs é a proliferação de diferentes formas de *spam* na Web. A seguir discutimos alguns trabalhos que evidenciam o uso de serviços encurtadores de URL para esse propósito.

Chhabra e colaboradores [8] provê uma ampla análise do espalhamento de *phishing* através de URLs encurtadas. Para essa análise, foram coletadas URLs de *phishing* contidas no site *PhishTank*[1], que é um site colaborativo de dados e informações sobre *phishing* na Internet. Depois de coletadas as URLs de *phishing*, um próximo passo foi encontrar as URLs curtas que redirecionavam para as URL encontradas anteriormente. Isso foi feito utilizando-se a API do Bit.ly, que possibilita a busca de todas URLs curtas de uma determinada URL longa e também fornece estatísticas de acesso das URLs. Uma análise interessante mostra que as URLs dos *phishers*, na sua maioria, tiveram pouca redução de tamanho em relação à suas respectivas versões longas ou até mesmo nenhum ganho, sugerindo que a função do uso de sistemas encurtadores por *phishers* é muitas vezes, para ofuscar as URLs maliciosas. Outra análise feita, mostra que websites de redes sociais como Twitter e Facebook competem com serviços de comércio eletrônico como PayPal[2] e eBay[3] em termos do foco de *phishers*.

Uma maneira de espalhar *spam* no Twitter, é através dos *trending topics*, que são assuntos mais falados na rede social. Os *spammers* criam *tweets* contendo palavras típicas do *trending topics*, mas com URLs que levam a sites que fogem completamente do assunto. Estes *tweets* postados normalmente contém URLs encurtadas, dificultando para os usuários identificarem o conteúdo da URL sem o carregamento da página. Relacionado a este assunto, Benevenuto e colaboradores [4] exploram uma maneira de detectar *spam* no Twitter. Para isso os autores fizeram uma coleta que contém uma coleção de $1,8$ bilhões de *tweets* e, em seguida, selecionaram *tweets* relacionados a três eventos que estiveram no *trending topics* (morte do Michael Jackson, aparecimento de Susan Boyle e

[1] ttp://www.phishtank.com/
[2] https://www.paypal.com/br/
[3] http://www.ebay.com/

*tweets* com "#musicmonday"). A partir desses *tweets*, foi construída uma grande coleção de usuários rotulados como *spammers* e *não-spammers*. Esses usuários rotulados foram utilizados pra identificar uma série de características que diferem o usuário *spammer* do *não-spammer*, essas características foram relacionadas a comportamento do usuário (ex.: número de seguidores, seguidos e tweets postados)e a conteúdo do *tweet* (ex.: fração de URL no *tweet* e *hashtags* por *tweet*). As características foram utilizadas como atributos em um processo de aprendizado de máquina para identificar os tipos usuários. Os resultados da estratégia foram satisfatórios, pois obtiveram sucesso na detecção de grande parte dos *spammers*. Aproximadamente 70% dos *spammers* e 96% de *não-spammers* foram corretamente classificados.

Em [22] é realizado um estudo sobre o uso de URLs curtas em uma escala global. Esse trabalho utilizou o serviço encurtador qr.cx [4] para realizar a coleta de URLs junto a com informações sobre a localização de sua criação, e onde foram utilizadas. Dessa grande coleta de URL curtas, foram retiradas aleatoriamente 5.957, das quais 80% foram identificadas como *spam*. Os autores criaram métricas para definir os países que mais criam ou resolvem URLs curtas (quem utiliza as URLs curtas). Foi descoberto que a utilização de serviços encurtadores difere entre os países. A criação de URLs curtas acontece em todo o mundo, mas são resolvidas principalmente na Europa e Estados Unidos, indicando que estes países tem uma maior tendência a ataque de *spammers*. Os autores também mostram que *spammers* seriam menos prováveis de verificar as URLs encurtadas e que os serviços encurtadores contribuem pra que ataque de *spam* cruzem fronteiras de outros países.

O trabalho [19] realiza uma caracterização de *spam* no Twitter. Para isso foi coletada uma amostra de dados do Twitter, onde foram examinados 400 milhões de *tweets*, dos quais foram retiradas 25 milhões de URLs únicas. Partindo dessas URLs e utilizando uma variedade de listas negras de URLs (Google Safebrowsing, URIBL e Joewein), foram encontradas 2 milhões de URLs com algum tipo de ataque aos usuários, isto representa 8% de todos os links postados no Twitter. Através desses *tweets* de *spam* que foram identificados, foi gerado uma lista dos termos mais frequentes, que em seguida foram classificados por categorias. A categoria que mais apareceu foi a de músicas grátis, games, livro e downloads gratuitos. Outra análise feita foi utilizando API do Bit.ly para coletar informações de clique sobre as URLs com *spam* que foram ofuscados pelo serviço de encurtamento Bit.ly, descobriu-se que o *spam* no Twitter é mais eficaz do que o *spam* no email. Cerca de $0,13\%$ dos *tweets* com *spam* gera uma visita à URL e para email

[4] http://qr.cx/

a taxa de visita à URL fica entre $0,003\%$ e $0,006\%$. Os autores também identificam alguns ataques de *spammers* utilizando encurtadores. Uma maneira é encurtar uma URL contendo *spam* em vários serviços diferentes, outra maneira identificada é o uso de encurtadores aninhados, onde uma URL que já está encurtada é encurtada novamente por outro serviço. Essas práticas têm como objetivo burlar a segurança tanto do próprio Twitter como dos serviços de encurtamento. Em uma última análise os autores provam a eficiência da integração de lista negra ao Twitter com o intuito de bloquear contas que postam URLs maliciosas. Através dessa análise foi identificado um atraso para as listas negras marcarem a URL no Twitter com *spam*, podendo variar de 4 a 20 dias. E como 90% dos acessos às URLs acontecem nos 2 primeiros dias após a postagem, essa integração ajuda uma pequena quantidade de usuários. Além disso, o uso de serviços encurtadores mascara às URLs, negando qualquer benefício potencial à integração de listas negras com o Twitter.

Goshi e colaboradores [16] realizaram um estudo sobre *link farming* no Twitter. *Link farming* é quando um usuário tenta adquirir influência na rede através da criação de seguidores falsos para si mesmo. Assim, quanto mais seguidores um usuário tiver, mais provável que seus *tweets* sejam altamente classificado em máquinas de busca. Para isso foi criada uma base de dados com contas de usuários *spammers*. Esta base foi criada através de contas que foram suspensas do Twitter por detectar alguma atividade maliciosa. Para identificar essas contas, os autores selecionaram todas as contas suspensas que continham URLs encurtadas pelos serviços Bit.ly e TinyURL, e em seguida era verificado a presença dessas URLs em listas negras. Com essa estratégia foi obtido 379.340 contas suspensas, das quais 41.352 postaram pelo menos uma URL encurtada que estava contida nas listas negra e utilizaram os serviços citados anteriormente. Com estes dados os autores confirmaram a existência de *link farming* no Twitter e mostraram um estratégia chamada CollusionRank, baseada no algoritmo do Trustrank [20] originalmente proposto para a Web. O objetivo CollusionRank é minimizar a atividade de *link farming* no Twitter.

Outra maneira de espalhar *spam* que vem sendo reportada em alguns trabalhos é o uso de robôs *bots*, que são programas de computador que controlam as contas de redes sociais e imitam usuários reais. O trabalho de Boshmaf e colaboradores [6], apresenta um estudo relacionado a vulnerabilidade do facebook para o uso de robôs. Para este estudo os autores criaram robôs no facebook e coletaram os dados referentes ao comportamento dos usuários onde os robôs infiltraram. Os resultados mostraram que os robôs conseguem infiltrar no facebook com uma taxa de sucesso de 80%. Outro trabalho que aborda o

assunto robôs é o de Messias e colaboradores [26], em que os autores criam robôs no Twitter para mostra que eles conseguem adquirir influência em sistemas de classificação de popularidade. Além de mostrar a facilidade de criação e manutenção de robôs no Twitter, eles também mostram que os sistemas Klout [5] e Twitalyzer [6] apresentam falhas em sua classificação, visto que eles conseguiram fazer com que um robô obtivesse altos índices de influências de acordo com esses sistemas.

Ainda no contexto de robôs, um estudo feito por Zi Chu e colaboradores [9] faz uma classificação nas contas de usuários, essa classificação é feita para mostrar quais contas são usuários humanos, robôs e *cyborgs*. Os *cyborgs* são usuários que se comportam às vezes como robô e às vezes como humano. Para essa classificação é feito uma coletada de dados do Twitter. E através desses dados os autores utilizam uma série de medidas para realizar a classificação das contas. Além das redes sociais, foram encontrados robôs em outros tipos de aplicação web, como chat online [18], blogs [33] e jogos on-line [17]. Um grande perigo que relaciona robôs às URLs curtas, é que depois de infiltrados nos sistemas, os robôs podem espalhar URLs com *spam* ofuscadas pelos encurtadores.

Os trabalhos mencionados acima apresentam evidências de spam no Twitter. Além do Twitter, spam tem sido observado em diferentes mídias sociais, como YouTube [3, 5], Facebook [15], Delicious [25], FourSquare [37], MySpace [23] e Apontador [11]. Potencialmente, encurtadores de URL também poderiam ser utilizados para ofuscar URLs nesses sistemas.

De maneira geral, os trabalhos sugerem que, apesar de úteis e extremamente populares, sistemas encurtadores estão sendo usados como uma forma de facilitar o espalhamento de URLs, das quais contém algum tipo de ameaça aos usuários das redes sociais e potencialmente a outras aplicações da web (ex.: email, comércio eletrônico). Outro ponto negativo é o atraso imposto, que é gerado com a ação de redirecionar para a URL original. Nossa contribuição nesse trabalho é investigar as bases para a construção de um sistema de encurtamento com uma arquitetura descentralizado que seja capaz de minimizar tais problemas.

[5] http://klout.com/
[6] http://twitalyzer.com/

# Capítulo 3

# Base de Dados

Para realizarmos nosso trabalho precisamos inicialmente de uma ampla coleção de URLs encurtadas postadas em sistemas sociais, como o Twitter. Mais importante, precisamos obter em grande quantidade a versão longa dessas URLs encontradas, de forma a permitir a avaliação da eficácia do mecanismo de compressão de URLs a ser proposto.

Nossa abordagem consiste em extrair URLs existentes em uma grande coleção de *tweets* obtida em um trabalho anterior [7] e depois traduzir essas URLs para suas versões originais. *Tweets* são as mensagens enviadas pelos usuário do Twitter. Essa coleção possui **54.981.152** usuários do Twitter, todos os elos de seguidores e seguidos (grafo com **1.963.263.821** arestas) e todos os *tweets* postados por esses usuários (**1.755.925.520** *tweets*). Esses *tweets* correspondem a todos os *tweets* já postados por todos os usuários do Twitter até o período da coleta, ou seja, desde a criação do Twitter em 2006 até julho de 2009. Essa base de dados é apropriada para o nosso propósito por se tratar de coleta completa do Twitter e não de uma amostra potencialmente tendenciosa. Para uma descrição mais detalhada desses dados e das técnicas empregadas para a realização de sua coleta, recomendamos ao leitor as seguintes referências [2, 7].

Visando traduzir, para suas versões longas, as URLs encurtadas encontradas nos *tweets*, foi desenvolvida uma ferramenta capaz de resolver à URL de um *tweet*. O sistema envia uma requisição de GET para cada URL e identifica o redirecionamento, que é o mecanismo utilizado por sistemas encurtadores. Ao detectar um redirecionamento, a versão longa original da URL é armazenada junto à versão curta. Esse procedimento foi realizado para todas as URLs encontradas na base e considerou-se que um serviço encurtador de URLs foi utilizado quando o domínio obtido era diferente e a URL era

maior do que a versão encurtada. Sendo assim, ao final desse procedimento foram obtidos diversos pares de URLs curtas e suas versões longas. No total identificamos **67.324.019** pares de URLs. Partindo desses pares de URLs, os domínios das URLs curtas foram ranqueados e os mais populares passaram por uma investigação manual, ou seja, forma abertos em um navegador, de forma que 77 diferentes encurtadores foram identificados. Esses encurtadores foram responsáveis pelo encurtamento de **57.763.943** (86% das URLs).

A tabela 3.1 mostra os 5 serviços encurtadores mais populares dentre os 77 encontrados. Esta tabela mostra as porcentagens e as quantidades de ocorrência dos serviços no ano de 2009, na base do Twitter. Para as análises apresentadas no restante do trabalho vamos utilizar os dois serviços mais populares, Bit.ly e TinyURL, que juntos representam mais do $87,24\%$, enquanto os outros 75 representam $12,76\%$.

| **Serviço** | **Porcentagem (%)** | **Ocorrência** |
|---|---|---|
| **Bit.ly** | **81,33** | **46.979.875** |
| **TinyURL** | **5,91** | **3.415.708** |
| Is.gd | 3,38 | 1.955.690 |
| Tweetburner | 2,87 | 1.662.796 |
| Ow.ly | 1,40 | 812.508 |
| Outros 72 | 5,08 | 2.937.366 |

**Tabela 3.1:** Serviços encurtadores de URL

A seguir abordamos duas características das URLs da base, a primeira característica está relacionada ao tamanho das URLs e a segunda análise o quando as URLs reduziram ao passar de longa para curta.

O gráfico da figura 3.1 mostra a distribuição de probabilidade cumulativa (CDF) do tamanho das URLs (número de caracteres). Visto que sistemas como o Twitter limitam o tamanho de mensagens em no máximo 140 caracteres, URLs com tamanhos na casa dos 100 caracteres praticamente se tornam inviáveis, visto que o espaço para o restante da mensagem seria muito curto. Entretanto, podemos notar que apenas 18% das URLs possuem tamanho maior que 100 e apenas 7% excedem os 140 caracteres.

O gráfico da figura 3.2 mostra a distribuição de probabilidade cumulativa (CDF) da taxa de encurtamento que os serviços Bit.ly e TinyURL obtiveram. Cabe ressaltar que a

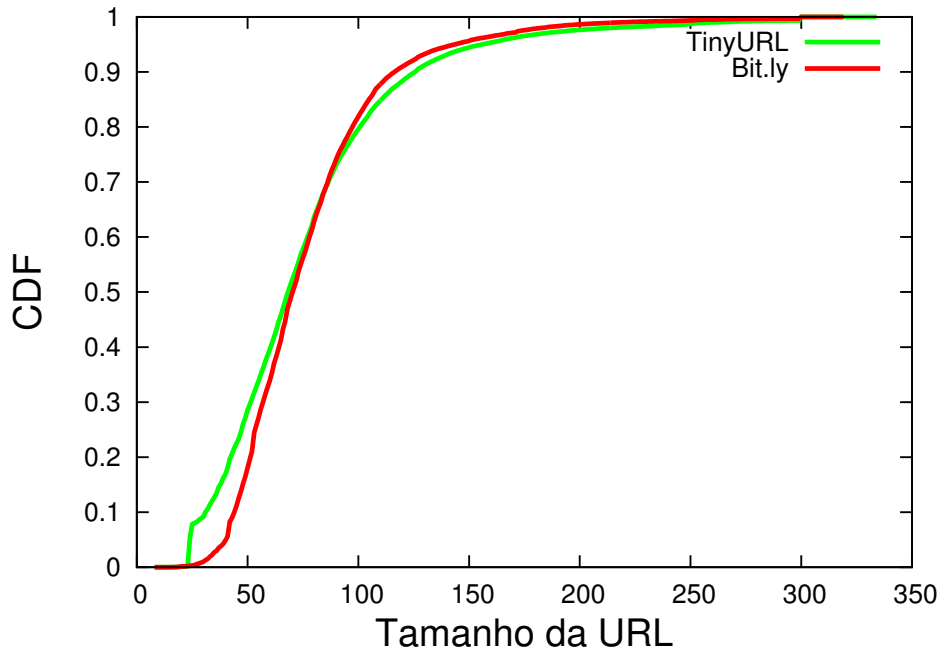



**Figura 3.1:** Distribuição cumulativa dos tamanhos das URLs nas bases do TinyURL e Bit.ly

compressão empregada por esses sistemas são ótimas, pois estes serviços utilizam tabelas *hash* para a criação das URLs encurtadas. Essa *hash* contêm tamanhos que variam de 3 a 6 caracteres, a variação de tamanho é devido ao esgotamento de combinações que formam as novas URLs. Cada caractere da *hash* pode ser A-Z, a-z e $0-9$, ou seja, são 62 caracteres, para uma *hash* de tamanho 3 seria possível encurtar 238.328 URLs. Porém mesmo com essa compressão ótima os encurtadores apresentam os problemas de uma arquitetura centralizada, os quais já foram mencionados no Capítulo 1.

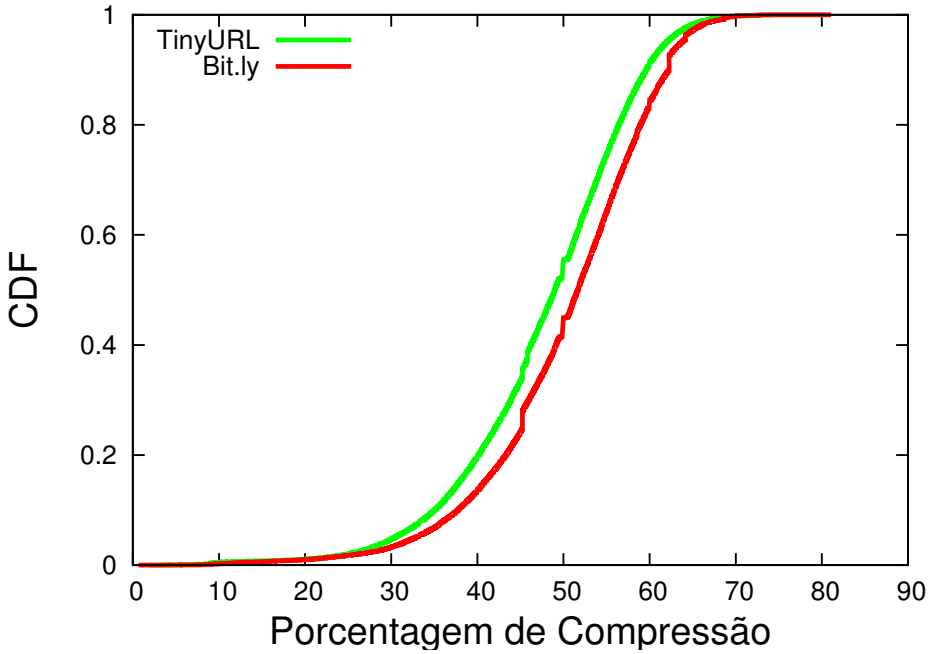



**Figura 3.2:** Taxas de encurtamentos obtidas pelo Bit.ly e TinyURL

# Capítulo 4

# Arcabouço do BeShort

Os sistemas encurtadores de URL atuais utilizam um arcabouço centralizada, como é ilustrado na figura 4.1. Nesses sistemas, usuários precisam encurtar suas URLs antes de serem postadas. Ao acessar uma URL encurtada, usuários fazem requisições para o serviço de encurtamento que retorna os códigos HTTP 301 ou 302 de redirecionamento para a URL longa original.

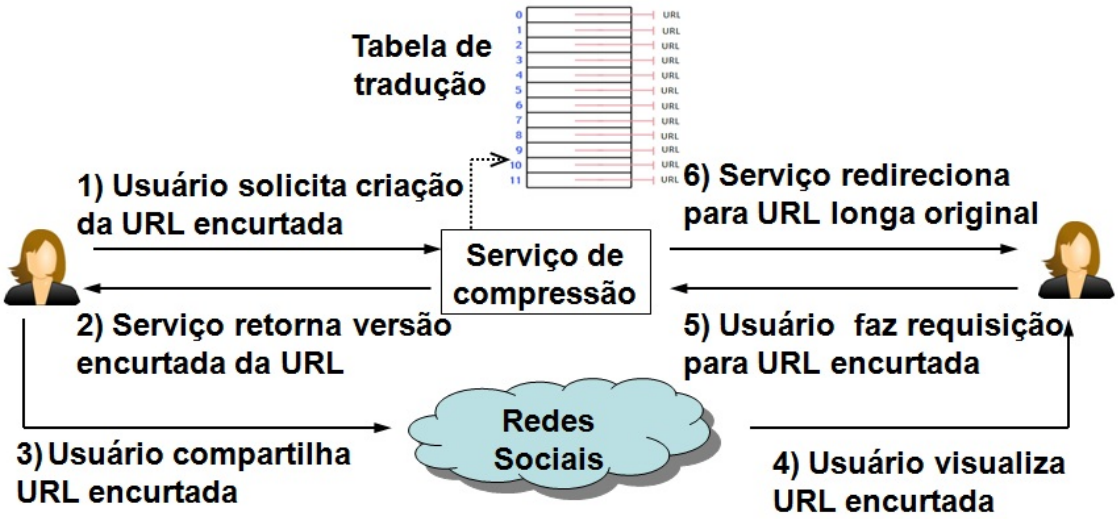



**Figura 4.1:** Arcabouço para encurtamento de URLs de forma centralizada

Nosso objetivo nesse trabalho é propor e avaliar a viabilidade de um arcabouço para encurtar URL que dispense o uso de um servidor central, de forma a evitar diversos aspectos indesejáveis do arcabouço centralizado. A figura 4.2 ilustra o funcionamento do arcabouço, que chamamos de BeShort. Na abordagem, usuários postam URLs em formato original. Essas, por sua vez, serão encurtadas pelo BeShort e em seguida enviadas de forma encurtada para as redes sociais. Mas antes da URL ser publicada, o BeShort será novamente acionado para realizar descompressão da URL para a sua forma original. Os procedimentos de compressão e descompressão do BeShort acontecem de forma transparente para os usuários que postam e para os que recebem.

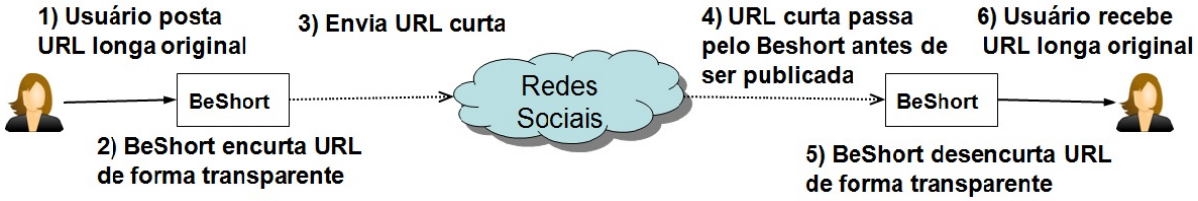



**Figura 4.2:** Arcabouço para encurtamento de URLs de forma descentralizada

Existem diferentes lugares onde o BeShort poderia ser implantado, como nos próprios clientes através de *plugins* em navegadores ou mesmo incorporado diretamente em aplicações de redes sociais específicas, como Twitter. Outra alternativa seria a implantação do BeShort em CDNs ( *content distribution networks*).

A estratégia de compressão do BeShort é baseada em substituir termos frequentes ocorridos em URLs (ex. http://www. ou google) por caracteres UTF normalmente não utilizados em URLs, porém aceitos em APIs de redes sociais online, como Twitter e Facebook. Para definir esses caracteres utilizamos o Padrão Unicode [35], que é o padrão de codificação de caractere universal utilizado na escrita de caracteres e textos. Existem três formas de codificação do Padrão Unicode que são: UTF-8, UTF-16 e UTF-32, onde qualquer uma delas pode representar toda a gama de caracteres do padrão, a diferença entre elas é a maneira de utilização. Para este trabalho iremos utilizar a forma UTF-8 pois, é a codificação mais utilizada no mundo na web [12].

Para permitir que uma URL do BeShort seja identificada e transformada novamente em sua versão longa original, é necessário uma forma de distinguir uma URL encurtada pelo BeShort, dos demais caracteres postados nas mensagens nas quais o BeShort foi compartilhado. No arcabouço, o identificador de uma URL encurtada pelo BeShort será o radical “b://”. A seguir será apresentado um exemplo de compressão de uma URL

utilizando o BeShort.

Suponha os termos e seus respectivos caracteres que estão sendo mostrados na tabela 4.1.

| **Termo** | **Caractere** |
|---|---|
| http:// | ♡ |
| www. | ∇ |
| decom | ⋈ |
| .br | Ψ |

**Tabela 4.1:** Exemplo ilustrativo da tabela de tradução de termos extraídos de URLs para caracteres do padrão UTF-8

Sendo assim, a URL http://www.decom.br, seria traduzida pelo BeShort para b://♡∇ ⋈ Ψ.

Finalmente, definir quais os termos devem ser substituídos por caracteres do padrão UTF-8 não é um tarefa trivial, visto que, termos de tamanhos diferentes podem possuir popularidades diferentes. Como exemplo, qual é a melhor estratégia, utilizar um caractere UTF-8 para o termo ".com" e outro para o termo ".br", ou devemos utilizar um único caractere UTF-8 para ".com.br"? A seguir, apresentamos nossa abordagem para a construção de um dicionário para tradução de URLs.

## 4.1 Construção do Dicionário

Para a construção do dicionário foram estudados alguns algoritmos de compressão e de criação de dicionários dinâmicos. Dentre eles, o que mais se aproxima do nosso interesse é o Algoritmo de Huffman baseado em palavras, que é eficaz em texto com linguagem natural [38]. Entretanto, o algoritmo de Huffman não é apropriado para a criação do dicionário, pois às URLs não possuem as mesmas características das linguagens naturais, não havendo, em virtude disso, como determinar um padrão claro de separadores que marcam o início e termino das palavras. Assim, não é possível determinar as palavras que estão contidas em uma URL. Por isso, decidiu-se desenvolver uma nova abordagem para a criação de um dicionário fixo.

O foco da criação do dicionário não foi em otimizar sua construção, mas em obter um melhor dicionário, um dicionário que tivesse termos que conseguissem melhores taxas de encurtamento das URLs. E cabe ressaltar que mesmo com abordagens exaustivas, o tempo de criação e a memória não foram problemáticos.

Nossa abordagem para a construção de um dicionário fixo é composta de três etapas importantes. Primeiramente, precisamos definir um tamanho para o dicionário, visto que existe um número limitado de caracteres no padrão Unicode que podem ser utilizados para a tradução de termos encontrados nas URLs (Seção 4.2). Em seguida precisamos gerar um grande número de termos (ex.: subpalavras existentes nas URLs) candidatos a fazer parte do dicionário. (Seção 4.3)Por fim, selecionamos os termos que farão parte do dicionário (Seção 4.4).

As próximas seções detalham nossas abordagens para cada uma dessas etapas.

## 4.2 Definição do Tamanho do Dicionário

O padrão Unicode contém **1.114.112** caracteres, dos quais os **65.536** correspondem aos caracteres utilizados nos principais idiomas do mundo e podem ser facilmente encontrados em bibliotecas de diferentes linguagens de programação [35]. As URLs comuns utilizam como caracteres apenas 128 dos 256 existentes no padrão ASCII, que não podem ser empregados na substituição dos termos do dicionário.

Desse modo, vamos considerar dois tamanhos de dicionários. O primeiro, que chamaremos de UTF-Parcial, considera como entrada do dicionário apenas os caracteres utilizados nos principais idiomas do mundo com exceção dos caracteres presentes no parão ASCII, contendo **65.408** caracteres. Note que o UTF-Parcial pode representar uma opção mais viável para implementação por consumir menos recursos (i.e. memória). O segundo, que será chamado de UTF-Total, corresponde a todas as possíveis entradas do padrão Unicode: **1.113.984**. Esses valores não correspondem aos mostrados anteriormente, pois, foram subtraídos os 128 caracteres utilizados em URLs, evitando, assim, que um determinado padrão não seja confundido com uma porção da URL que não foi traduzida.

## 4.3 Definição dos Termos Candidatos do Dicionário

Para a criação dos termos candidatos, foi definido um algoritmo que recebe como entrada um conjunto de URLs, denominado conjunto de treino $T$. Em seguida, fazemos a extração de todos os potenciais termos (subpalavras) das URLs de $T$, com tamanho $i$ $|2 \leq i \leq M$, onde $M$ é o valor máximo para o tamanho de um termo, definido empiricamente e discutido no Capítulo 5, Seção 5.5. O tamanho começa em 2, pois é o menor tamanho em que ainda se pode ter um ganho na compressão. Assim, para cada valor de $i$, o algoritmo extrai todos termos com esse tamanho da URL $u$, retirada da base de treino $T$.

Ao percorrer o conjunto $T$, a frequência de ocorrência dos termos identificados é contabilizada. Os termos identificados e suas frequências são armazenados no conjunto $S$. Após ter gerado todos os termos e contabilizado todas as frequências, é realizado uma ação com os termos do conjunto $S$. Esta ação é realizada para gerar o conjunto final $F$, que contém os termos e suas respectivas características, que são, frequência, tamanho (número de caracteres) e a multiplicação da frequência pelo tamanho do termo. Os termos das URLs gerados em $F$ são considerados termos candidatos a formarem o dicionário. Com isso o conjunto $F$ de saída do nosso algoritmo será utilizado pelas diferentes estratégias de seleção discutidas na Seção 4.4. Os passos para a construção dos termos candidatos estão descritos no Algoritmo 4.1.

**Algoritmo 4.1**: Algoritmo para criação dos termos candidatos ao dicionário

```
   Entrada: Entrada: Conjunto T de URLs;
   Saída: Conjunto F de termos com suas características
 1 tamanho do termo i = 2;
 2 enquanto i < M faça
 3 |   enquanto existir URL u em T faça
 4 |   |   enquanto u não atingir o fim faça
 5 |   |   |   Para cada termo s de tamanho i em u;
 6 |   |   |   se s ∉ S então
 7 |   |   |   |   Insere s em S e inicia o valor da frequência de s igual a 1;
 8 |   |   |   senão
 9 |   |   |   |   Soma 1 à frequência de s;
10 enquanto existir termo s em S faça
11 |   Insere s em F, junto a sua frequência, tamanho e frequência × tamanho;
```

Na tabela 4.2 é mostrado um exemplo de como ficaria o conjunto final $F$ com todos

os possíveis termos candidatos, onde na primeira coluna temos os termos gerados, na segunda a quantidade de vezes que eles apareceram, a terceira coluna o tamanho dos termos e a quarta coluna a multiplicação das duas colunas anteriores.

| **Termo** | **Frequência** | Tamanho | Frequência × Tamanho |
|---|---|---|---|
| http://www. | 100 | 11 | 1100 |
| .com.br | 150 | 7 | 750 |
| google | 90 | 6 | 540 |
| ufop | 300 | 4 | 1200 |
| youtube.br | 200 | 10 | 2000 |
| glo | 100 | 3 | 300 |
| ⋮ | ⋮ | ⋮ | ⋮ |

**Tabela 4.2:** Conjunto $F$ de termos candidatos

## 4.4 Estratégias para Seleção de Termos

Após definido a quantidade de termos que irão compor o dicionário e ter gerado o conjunto $F$ de termos candidatos na segunda etapa, precisamos selecionar os termos que efetivamente farão parte do dicionário. A seguir são apresentadas cinco estratégias para seleção desses termos.

1. **Aleatório** - Os termos são selecionados de $F$ de forma aleatória.
2. **Tamanho** - São selecionados o termos de $F$ que tiverem maior tamanho.
3. **Frequência X Tamanho** ($Freq \times Tam$) - A seleção é feita com os termos de $F$ que tiverem o maior resultado da operação $frequencia \times tamanho$.
4. **Frequência** - Os termos de $F$ que tiverem os valores mais altos de frequência são selecionados.
5. **Frequência menos Subpalavra** ($Freq - Sub$) - Primeiro o conjunto $F$ é ordenado em ordem decrescente da frequência dos termos. Em seguida, excluímos os termos de menor frequência que são subpalavras de termos com maiores frequências. Como

exemplo, se o termo "http://www"é mais frequente que o termo "http://", esse último não é incluído no dicionário visto que ele é uma subpalavra do primeiro e é menos frequente.

A seleção de termos é feita até que alcance os tamanhos de dicionário definidos na Seção 4.2.

# Capítulo 5

# Avaliação Experimental

Neste capítulo apresentamos uma avaliação de desempenho do BeShort. Inicialmente, na Seção 5.2 fazemos uma análise das estratégias de seleção de termos para o dicionário. Posteriormente, a Seção 5.3 mostra os principais resultados relacionados a eficiência da porcentagem de compressão. Na Seção 5.4 estudamos o impacto do tamanho da URL na compressão. Em seguida, a Seção 5.5 explora parâmetros da construção do dicionário do BeShort. Por fim, na Seção 5.6 calculamos os atrasos impostos pelos serviços encurtadores.

## 5.1 Ambiente Experimental

Para a realização dos experimentos utilizamos duas bases, cada uma contendo 1 milhão de URLs dos serviços Bit.ly e TinyURL, que foram obtidas aleatoriamente de uma base de dados que contém todas às URLs extraídas dos *tweets*. Para construir o dicionário utilizamos 500 mil URLs do sistema do Bit.ly e outras 500 mil do sistema do TinyURL. A construção do dicionário foi utilizando uma máquina Linux com um processador AMD Phenom II e 4 GB de memória.

Alguns resultados utilizam o termo "porcentagem de encurtamento", que representa a porcentagem de caracteres que diminui da URL longa quando esta é encurtada. Cabe ressaltar que nessas porcentagens de encurtamento, foi considerada a parte fixa da URL encurtada (ex. b://, www.bit.ly/, www.tinyurl.com/).

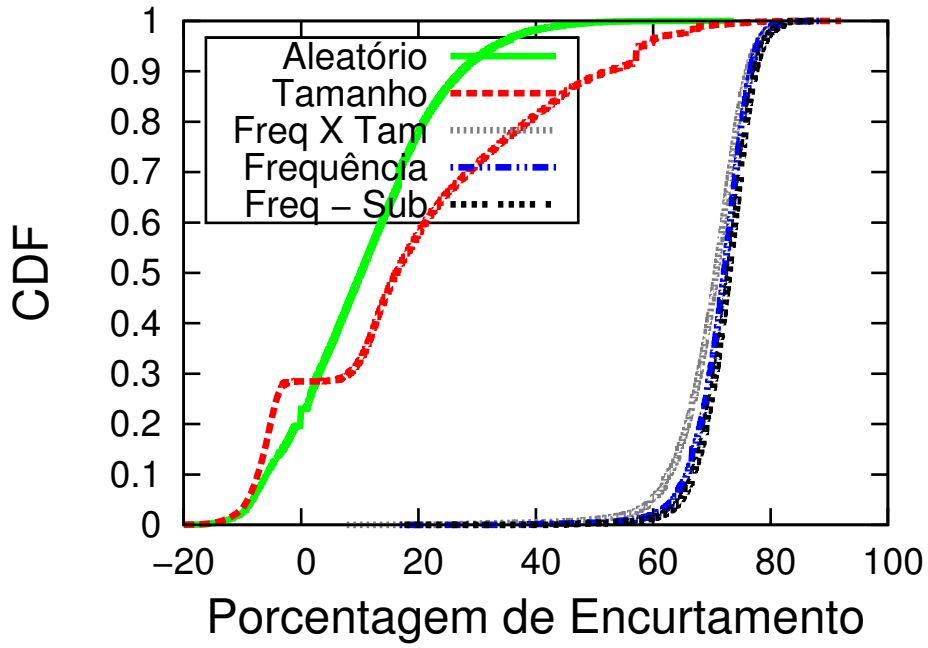
## 5.2 Análise das Estratégias de Seleção dos Termos

As comparações entre as estratégias foram feitas em função da porcentagem de encurtamento alcançada para cada URL. Para gerar as porcentagens, selecionamos 1 milhão de URLs longas de forma aleatória, elas foram retiradas das bases do Bit.ly e do TinyURL. Os resultados do experimento são mostrados nos gráficos da figura 5.1 onde podemos visualizar a distribuição de probabilidade cumulativa (CDF) das porcentagens de encurtamento de cada uma dessas estratégias.

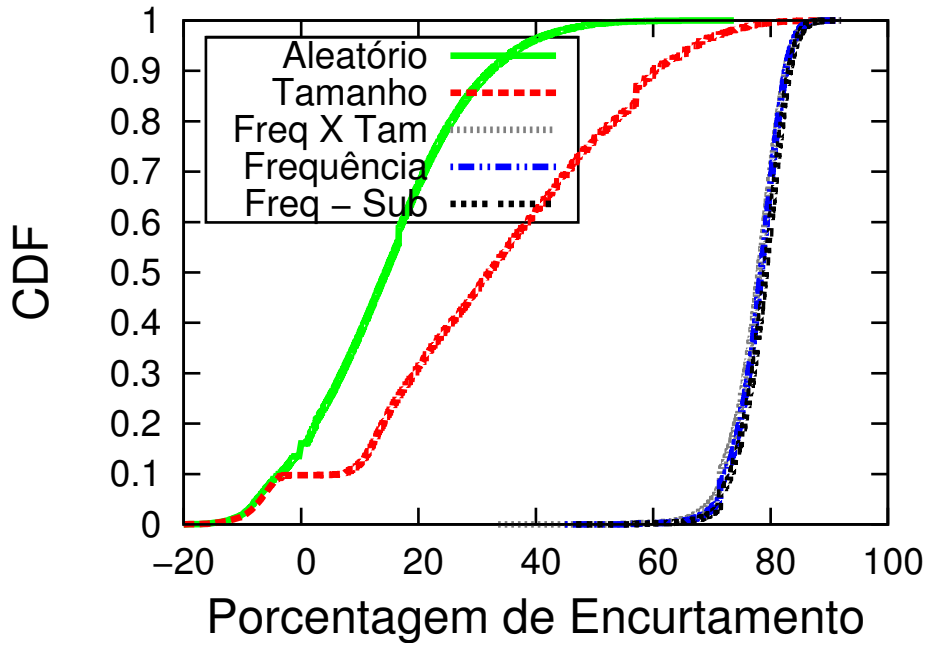
Analisando o gráfico da figura 5.1(a), no qual foi utilizado o tamanho de dicionário UTF-Parcial, percebemos que as estratégias *Aleatório* e $Tamanho$ foram bem inferiores se comparadas às outras. Como exemplo, enquanto 98% das URLs obtiveram mais que 60% de encurtamento para a estratégia $Frequencia$, apenas $0,1\%$ e 4% das URLs conseguiram taxas de encurtamento superiores aos mesmos 60% para as estratégia *Aleatório* e $Tamanho$. Comparando as estratégias $Freq \times Tam$, *Frequência* e $Freq{-}Sub$, observamos que elas estão bem próximas porém, a estratégia $Freq{-}Sub$ possui um ganho marginal em relação às as outras. Como exemplo, 80% das URLs obtiveram taxas de encurtamento superiores a 69% para estratégia $Freq{-}Sub$, enquanto que nas estratégias $Freq \times Tam$ e *Frequência*, 80% das URLs obtiveram taxas de encurtamento superiores a 66% e 68%, respectivamente. Analisando a média das porcentagens de encurtamento, percebe-se que a estratégia $Freq{-}Sub$ foi superior às outras com valor de $72,40\%$, enquanto a estratégia $Freq \times Tam$ foi $69,81\%$ e a estratégia *Frequência* foi $71,49\%$.

No gráfico da figura 5.1(b), foi utilizado o tamanho de dicionário UTF-Total, neste gráfico podemos tirar as mesmas conclusões anteriores, e reforçar que a estratégia é a $Freq{-}Sub$ é superior as demais.

Analisados os dois gráficos, podemos concluir que a estratégia que obteve melhores taxas de encurtamento nas URLs foi a **Freq−Sub**, e por este motivo iremos utilizá-la na criação do dicionário para os próximos experimentos.




(a) UTF-Parcial




(b) UTF-Total

**Figura 5.1:** Compressão das estratégias de seleção dos termos

## 5.3 Análise de Compressão

Nesta seção o BeShort será testado em relação a porcentagem de encurtamento das URLs, e seus resultados serão comparados aos serviços Bit.ly e TinyURL. Os gráficos das figuras 5.2 e 5.3 mostram a distribuição de probabilidade cumulativa (CDF) da porcentagem de encurtamento de cada um desses serviços. O gráfico da figura 5.2 compara o BeShort com o Bit.ly, sendo que para o BeShort foi utilizado os dois tamanhos de dicionário anteriormente discutidos: UTF-Total e UTF-Parcial. Apesar das três curvas estarem próximas e se cruzarem, podemos observar que as três abordagens conseguem resultados de compressão bons e competitivos. Boa parte das URLs dos três sistemas obtiveram porcentagens de encurtamento entre 60% e 80% de encurtamento. Podemos notar que em alguns casos, o BeShort é superior ao Bit.ly. Como exemplo, cerca de 80% das URLs obtiveram taxas de encurtamento superiores a 75% com o dicionário UTF-Total, enquanto que apenas cerca de 40% das URLs encurtadas com o Bit.ly conseguiram porcentagens tão altas.

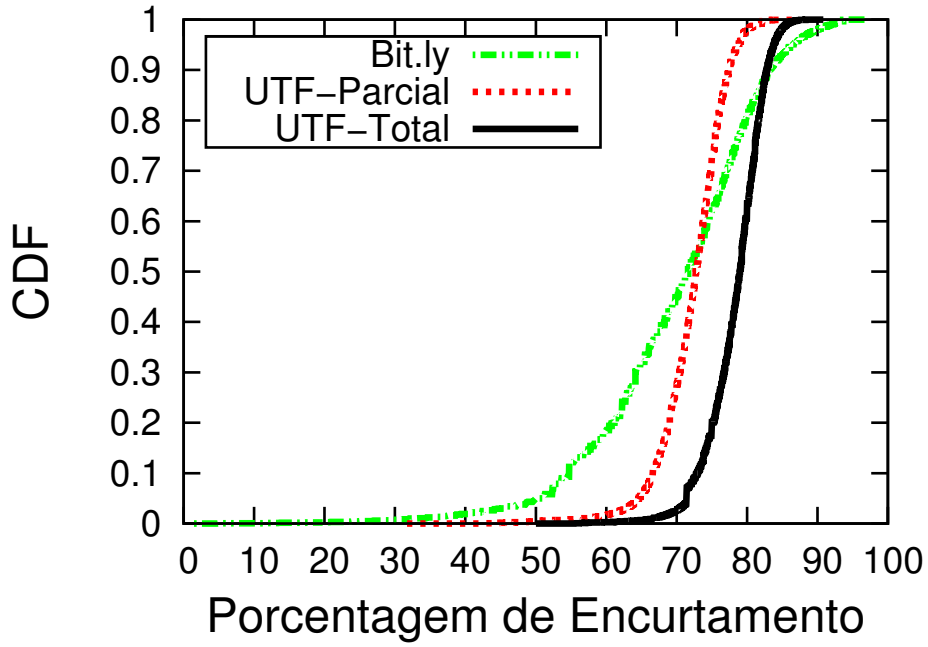



**Figura 5.2:** Compressão do BeShort na base do Bit.ly

A competitividade do BeShort com as arquiteturas centralizadas fica ainda mais evidente na comparação com o TinyURL. O gráfico da figura 5.3 apresenta a mesma análise, porém utiliza a base de URLs do TinyURL e mostra uma comparação com o encurtamento desse serviço. Enquanto 90% das URLs encurtadas com o BeShort

utilizando UTF-Total obtiveram porcentagens de encurtamento superiores a 73%, ao passo que apenas 25% dos encurtamentos do TinyURL conseguiram porcentagens acima desse valor.

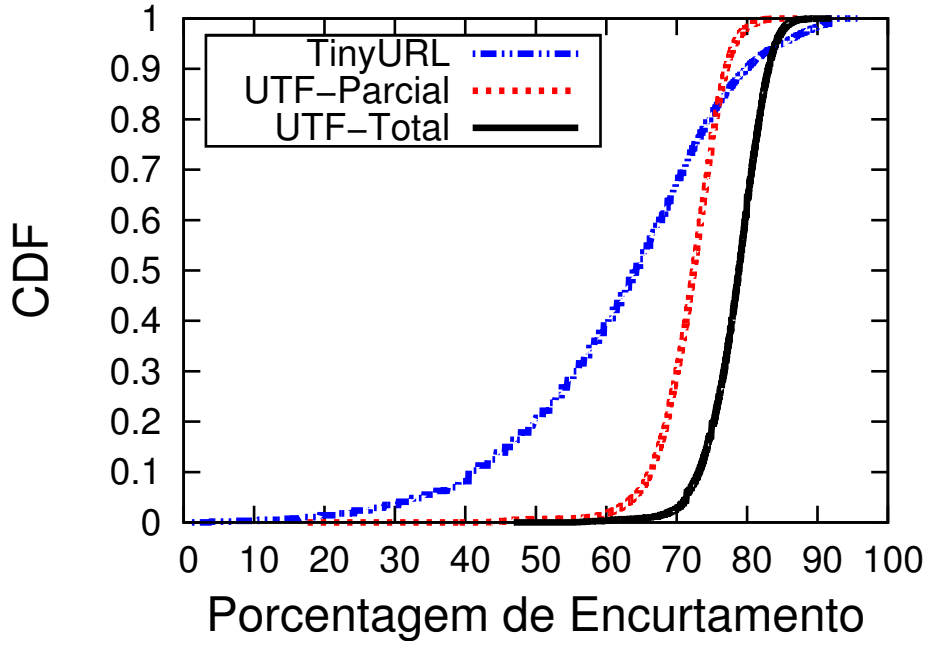



**Figura 5.3:** Compressão do BeShort na base do TinyURL

Comparando os resultados do encurtamento do BeShort com o dicionário UTF-Parcial e UTF-Total, podemos notar que o UTF-Total é melhor do que o UTF-Parcial, porém ambos são competitivos. A vantagem do uso do UTF-Parcial é que ele pode simplificar a implantação do BeShort por utilizar caracteres UTF-8 normalmente suportados por bibliotecas de linguagens de programação e normalmente aceitas em navegadores e outros programas. Além disso, o UTF-Parcial reduz drasticamente a quantidade de memória necessária, o que pode ser essencial para o caso de implementar o BeShort em dispositivos móveis.

Em sumário, os resultados dessa seção mostram que o BeShort é superior ao Bit.ly e TinyURL na maioria dos casos, e que o BeShort com o dicionário UTF-Total alcança resultados melhores comparados aos resultados do BeShort com dicionário UTF-Parcial.

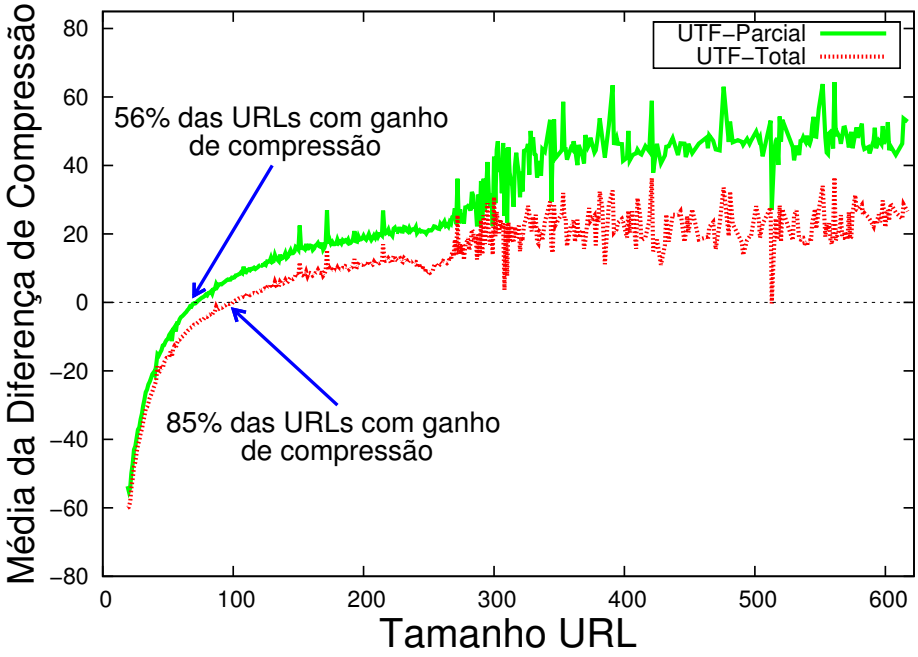
## 5.4 Impacto do Tamanho da URL

Com o intuito de analisar como o tamanho da URL pode afetar as porcentagens de encurtamento do BeShort e dos serviços Bit.ly e TinyURL, realizamos a seguinte análise. Para cada tamanho $X$ das URLs longas da nossa base de testes, calculamos a média de compressão para este tamanho. Em outras palavras, nós calculamos a porcentagem de encurtamento segundo BeShort UTF-Total, BeShort UTF-Parcial, Bit.ly e TinyURL para às URLs da base, agrupamos por seus tamanhos e dividimos pelo número de URLs existentes com cada tamanho.

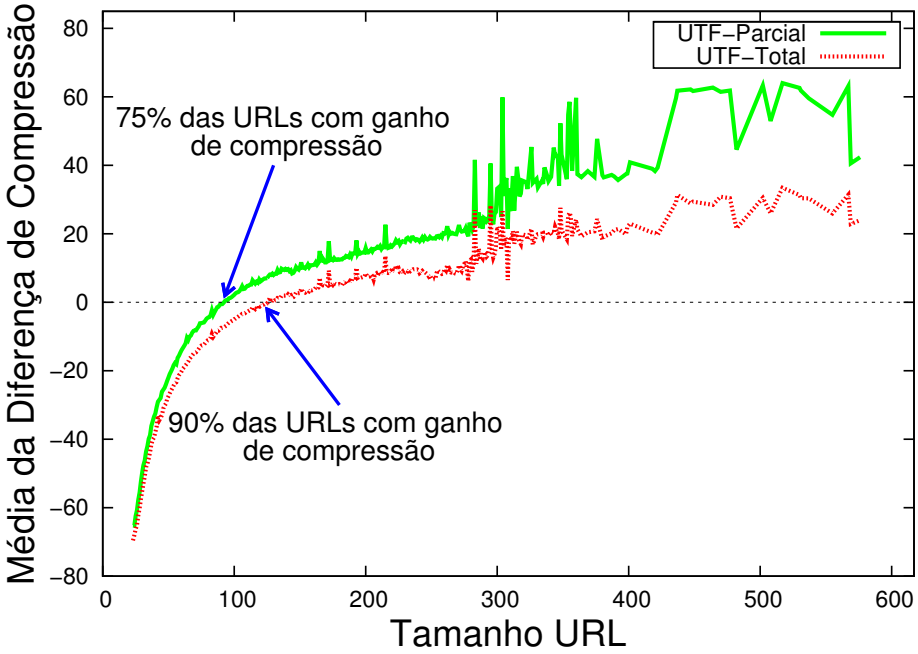
Os gráficos da figura 5.4 mostram a diferença das médias de compressão em função do tamanho das URLs longas. Comparando o BeShort com os outros serviços. Quando o resultado é um valor negativo significa que o BeShort obteve um encurtamento melhor e quando o valor é positivo significa que os serviços, Bit.ly ou TinyURL alcançaram melhores resultados.

O gráfico da figura 5.4(a) apresenta uma comparação da diferença das médias do Bit.ly com o Beshort, utilizando os dois tamanhos de dicionário. Com o dicionário UTF-Total, o BeShort perde para URLs com tamanho superior a 100 caracteres, por outro lado o UTF-Parcial começa a perder para o Bit.ly a partir de URLs com 72 caracteres. Analisando o segundo gráfico da figura 5.4, mostramos a mesma comparação, mas agora entre o TinyURL e o BeShort, que está utilizando os mesmos dois tamanhos de dicionários. Pode-se perceber que, com o UTF-Total o BeShort não obteve resultados satisfatórios na compressão para URLs de tamanho acima de 125 caracteres, já no UTF-Parcial este valor cai para 93 caracteres.

Com as análises anteriores podemos notar que o BeShort é, em geral, mais efetivo do que os serviços Bit.ly e TinyURL para URLs de tamanho menor. Também é importante ressaltar que, mesmo apresentando resultados piores para URLs com um número elevado de caracteres, o BeShort ainda consegue resultados competitivos comparando-o com as arquiteturas centralizadas. De maneira geral, nos casos em que o BeShort perde, as diferenças das médias ficam em sua maioria na casa dos 20%, o que ainda torna o BeShort viável. Além disso, foi observado em nossas análises que, em média 18% das URLs possuem tamanho maior que 100 e apenas 7% excedem os 140 caracteres. Em uma breve inspeção manual, analisamos 50 URLs com tamanhos superiores a 100 caracteres e notamos que elas, em geral, correspondem a endereços contendo mapas ou informações relativas a sessões de usuários.




(a) Diferença para o Bit.ly




(b) Diferença para o TinyURL

**Figura 5.4:** Diferença de compressão entre BeShort e os demais serviços

## 5.5 Tamanho Máximo dos Termos

Finalmente, um parâmetro importante do nosso algoritmo para a criação do dicionário é o tamanho máximo dos termos utilizados. Os três gráficos mostrados nas figuras 5.5 e 5.6 oferecem análises referentes a esse parâmetro, e medem o quanto o tamanho do termo pode influenciar na porcentagem de encurtamento. O gráfico da figura 5.5 mostra a média de encurtamento obtida para todas às URLs da base em função do tamanho máximo do termo. O primeiro tamanho máximo avaliado começa com 4 pois, com os tamanhos 2 e 3 não foi possível criar um dicionário com o número de termos que pudesse preencher o dicionário com UTF-Total, ou seja, **1.114.112** termos. Pode-se observar que a média de compressão é crescente entre os tamanhos máximos 4 e 8, e após este valor a média começa a estabilizar.

De fato, os gráficos das figuras 5.6(a) e 5.6(b) mostram a distribuição de probabilidade cumulativa (CDF) da porcentagem de encurtamento variando o tamanho do termo na construção do dicionário do BeShort. Tanto no UTF-Parcial quanto no UTF-Total podemos perceber que as curvas estão muito próximas para os valores 10 e 15.

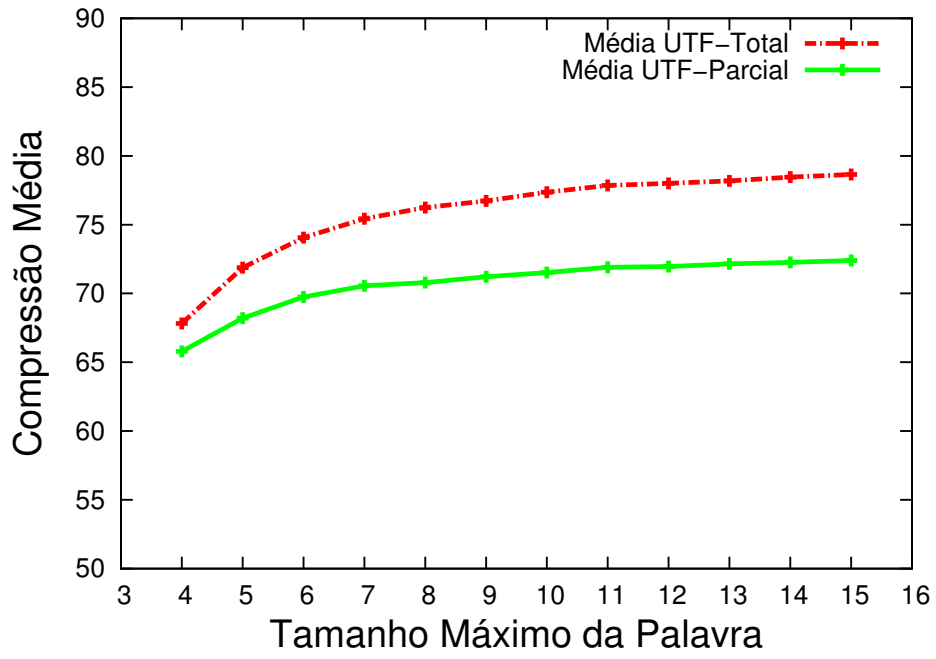
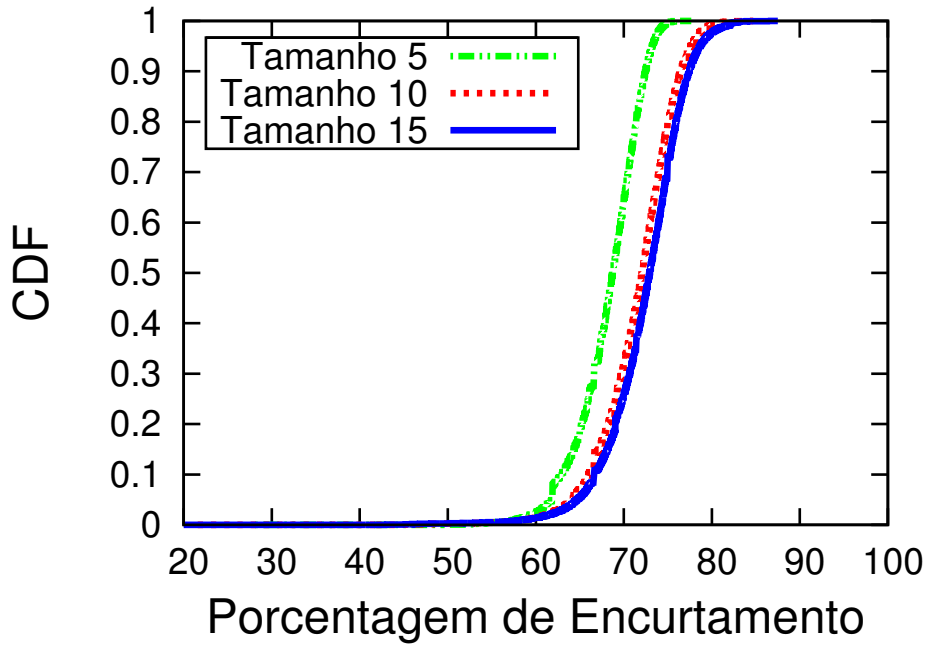
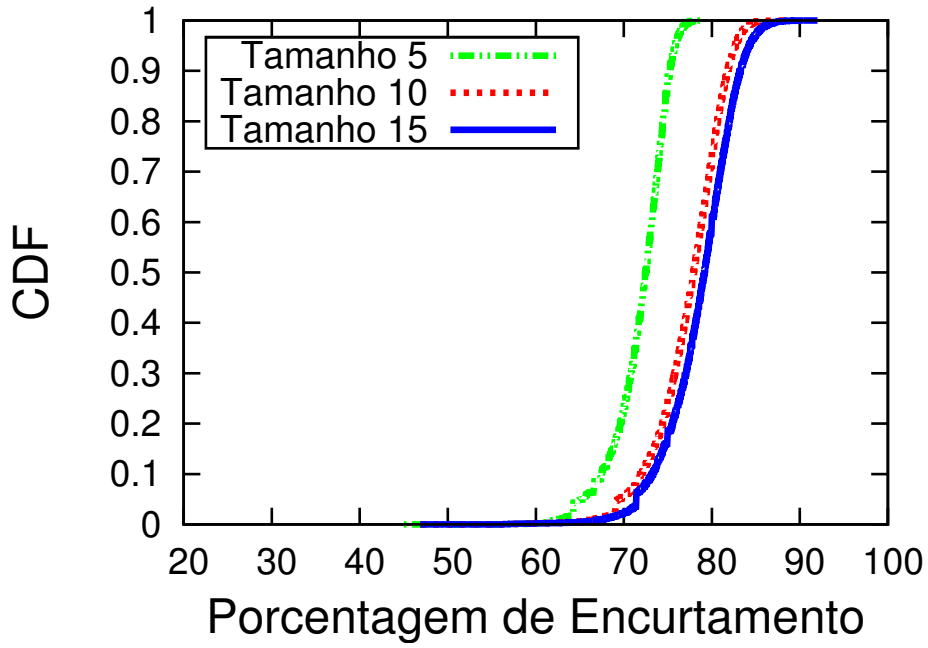



**Figura 5.5:** Média de compressão à medida que varia o tamanho máximo do termo

A partir das análises mencionadas, podemos concluir que se o tamanho máximo




(a) UTF-Parcial




(b) UTF-Total

**Figura 5.6:** Compressão em função do tamanho máximo do termo

do termo for próximo a 8 já é suficiente para obter resultados tão precisos quanto os apresentados nas seções anteriores, calculados com tamanho máximo igual a 15.

## 5.6 Atrasos Impostos pelos Serviços

Um problema encontrado nos encurtadores atuais é o atraso causado durante o redirecionamento da URL encurtada para sua versão longa. No caso do BeShort, ações que podem causar algum atraso em seu funcionamento são as operações de encurtar e desencurtar às URLs. Nesse contexto, a seguir medimos o atraso imposto pelos serviços Bit.ly e TinyURL para verificar se podem ser significativos (Seção 5.6.1). Em seguida, comparamos esse atraso com possíveis atrasos que podem ocorrer no encurtamento e desencurtamento do BeShort (Seção 5.6.2).

Para estimar estes tempos de atraso foram separados, de forma aleatória, 2000 pares de URLs, sendo 1000 do serviço Bit.ly e 1000 do TinyURL. Assim, para cada serviço foram obtidas 2000 URLs, 1000 curtas e 1000 longas. Estes pares foram retirados da mesma base mencionada no Capítulo 3.

### 5.6.1 Atraso no Redirecionamento

Para medir o atraso imposto no redirecionamento pelos serviços Bit.ly e TinyURL, cada URL, tanto curta quanto longa, foi acessada 4 vezes por dia, durante 10 dias. Para cada acesso à URL, registramos o tempo total de transferência da página. Após ter obtido todos os tempos de acesso para às URLs, foi calculada a média destes tempos. Estes acessos às URLs foram efetuados de uma máquina situada na Universidade Federal de Ouro Preto.

No gráfico da figura 5.7 mostramos uma distribuição de probabilidade cumulativa (CDF), que é resultante da razão obtida entre a média do tempo de acesso da URL curta pela URL longa, para o Bit.ly e TinyURL. Nele o eixo $x$ representa quantas vezes o tempo para acessar a URL encurtada aumentou em relação à URL longa original. Podemos observar que houve um aumento no tempo de acesso para os dois serviços, e que o Bit.ly obteve um aumento menor comparado ao TinyURL. No Bit.ly aproximadamente 94% dos acessos obtiveram uma razão inferior a 2, ou seja, dobraram o tempo, já no TinyURL os acessos que obtiveram a mesma marca aproximaram-se de 60%.

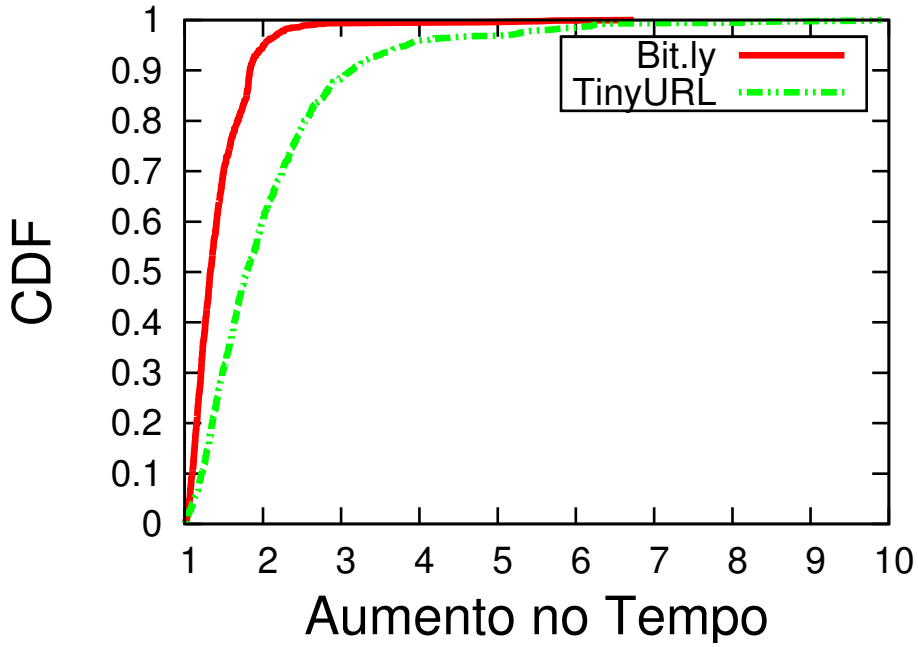



**Figura 5.7:** Razão do aumento do tempo de acesso para os serviços Bit.ly e TinyURL

Um segundo teste foi em relação ao tempo de redirecionamento, que é o tempo que a URL curta demora para retornar a URL longa. Nele foram utilizados os mesmos 1000 pares de URLs, mas usamos somente às URLs curtas. O teste foi feito a partir da ferramente desenvolvida para resolver a URL curta (ferramenta mencionada no Capítulo 3) e teve duração de 10 dias, sendo que, a cada dia foram contabilizados 4 tempos de redirecionamento para cada URL curta. Os tempos de redirecionamento foram considerados como sendo o intervalo para a ferramenta retornar a URL longa.

Visualizando o gráfico da figura 5.8, em que é apresentada uma distribuição de probabilidade cumulativa (CDF) do tempo gasto (em segundos) no redirecionamento das URLs curtas, percebemos que no Bit.ly 68% das URLs gastaram no máximo 2 segundos pra realizar tal tarefa, ao passo que no TinyURL as mesmas 68% das URLs gastaram, no máximo, $2,32$ segundos.

Após as análises dos testes mencionados, podemos notar que o atraso encontrado pode ser significativo visto que sua média foi de $1,79$ segundos para o Bit.ly e $2,10$ segundos para o TinyURL. Além deste atraso ser causado pela localização do servidor e do usuário que realiza a requisição, outras consequências podem aumentar esta latência. Um exemplo seria a sobrecarga do servidor, já que estes serviços estão cada dia mais

populares, com isso pode acarretar em um aumento do número de requisições.

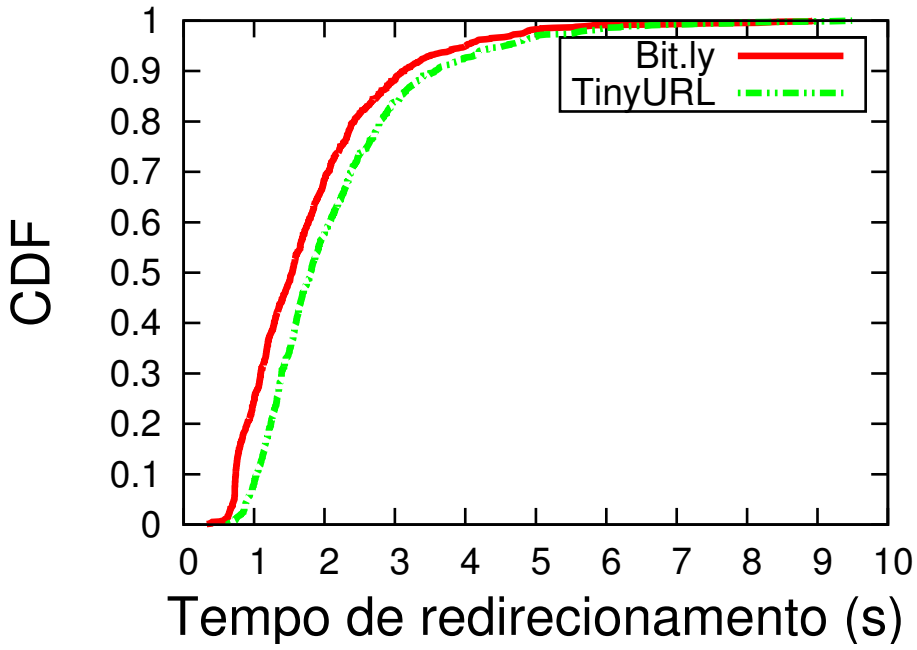



**Figura 5.8:** Tempo em segundos para o redirecionamento

### 5.6.2 Tempo Gasto para Realizar as Operações de Encurtar e Desencuratar às URLs pelo BeShort

A seguir comparamos o tempo gasto de encurtamento e desencurtamento do BeShort com o tempo de redirecionamento imposto pelo Bit.ly e TinyURL. Para esta análise utilizamos somente às URLs longas mencionadas anteriormente. Para cada URL longa nos calculamos o tempo necessário para a realização das operações de encurtar e desencurtar a URL, este procedimento foi realizado 10 vezes. Em seguida, retiramos o maior e o menor tempo medidos para evitar distorções nos resultados e computamos a média dos tempos obtidos a partir das oito medidas restantes. Está análise é realizada com o BeShort tendo o dicionário de tamanho UTF-Parcial.

No gráfico da figura 5.9 apresentamos uma distribuição de probabilidade cumulativa (CDF) dos tempos obtidos anteriormente para às URLs dos dois serviços, Bit.ly e TinyURL. Nele, podemos observar que 90% das URLs do Bit.ly gastaram no máximo $0,000120$ segundos para serem encurtadas e desencurtadas. De forma semelhante, no TinyURL 90% das URLs alcançaram no máximo $0,000147$ segundos. Em média, os dois

serviços Bit.ly e TinyURL gastaram, respectivamente, $0,00006574$ e $0,00008416$ segundos. Com esses resultados podemos perceber que o tempo para a realização das ações de encurtar e desencurtar é muito pequeno, pois equivale a uma fração pequena de um segundo.

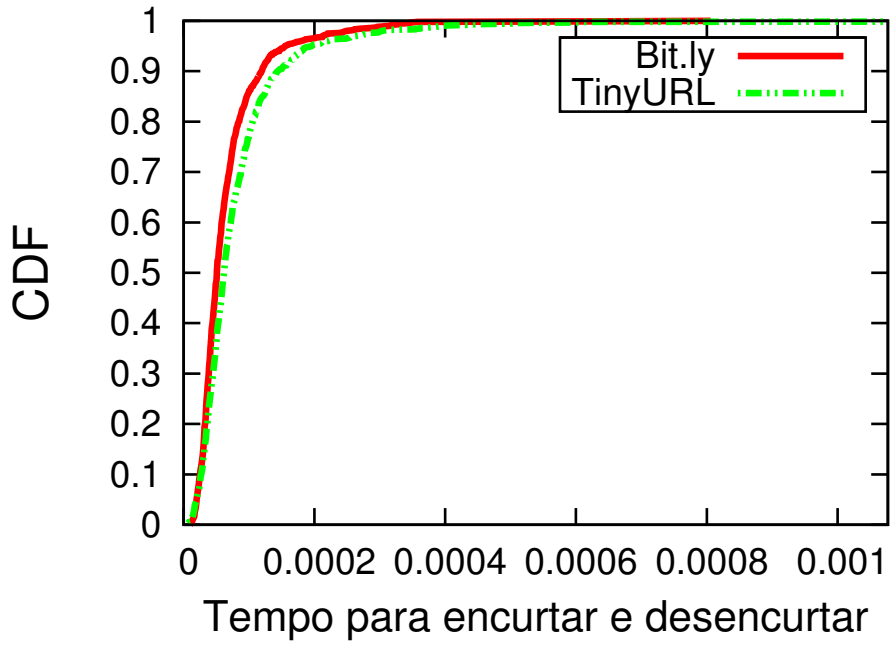



**Figura 5.9:** Tempo em segundos para as ações de encurtar e desencurtar às URLs realizadas pelo BeShort

Finalizando as análises em relação aos atrasos impostos pelos serviços Bit.ly, TinyURL e BeShort, nós comparamos o atraso relativo imposto por cada abordagem de encurtamento de URLs. No gráfico da figura 5.10 exibimos uma distribuição de probabilidade cumulativa (CDF) da razão entre o tempo de atraso causado pelo redirecionamento dos serviços Bit.ly e TinyURL pelo tempo de atraso causado pelas funções de encurtar e desencurtar às URLs do BeShort. Podemos observar que para o serviço Bit.ly 90% dos tempos tiveram uma razão superior a 10.238, já para o TinyURL este número foi um pouco maior, onde 90% dos tempos alcançaram razões acima de 10.467.

Com essas observações podemos concluir que o atraso imposto pelo redirecionamento realizado por arquiteturas centralizadas como as do Bit.ly e TinyURL é muito maior que o tempo gasto para concluir as operações de encurtar e desencurtar a URL realizadas pelo BeShort.

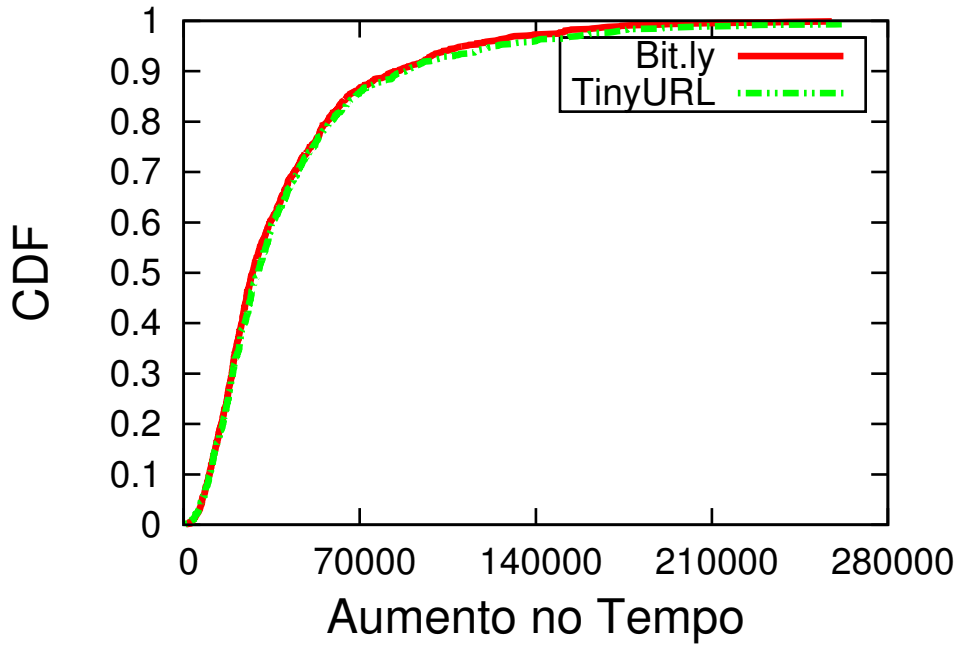



**Figura 5.10:** Razão entre o atraso imposto pelos serviços Bit.ly e TinyURL sobre o atraso imposto pelo BeShort

# Capítulo 6

# Protótipo do BeShort

Como forma de apresentar uma prova de conceito do funcionamento do BeShort, nós implementamos seu protótipo. Neste capítulo apresentamos detalhes dessa implementação e alguns testes realizados. Com mencionamos anteriormente, o BeShort pode ser implantado de diversas maneiras, e uma delas é através de conexão com as redes sociais, onde a conexão é feita por meio da API. Com isso fizemos uma ferramenta que estabelece conexão com o Twitter, pela sua API[1]. Utilizamos a API do Twitter com o auxílio da biblioteca Twitter4j[2], que é um biblioteca Java que integra com maior facilidade uma aplicação com os serviço oferecidos pela API. A ferramenta foi desenvolvida utilizando a tecnologia JavaServer Pages (JSP). O código deste protótipo esta disponível em `https://github.com/pedropufop/beshort`.

Um primeiro passo para implementar a ferramenta foi a criação do dicionário com os termos e as funções de encurtar e desencurtar às URLs. Um outro passo consiste em integrar o BeShort à API do Twitter. Para realizar tal integração registramos o BeShort junto ao Twitter, para que o Twitter identifique a aplicação e, consequentemente, autorize seu acesso através da sua API. Além do registro da aplicação, é necessário que contas do Twitter autorizem o BeShort a manipularem seus dados. Com isso criamos duas contas no Twitter: @BeShort2012_1[3] e @BeShort2012_2[4], e em seguida foi concedido a autorização ao BeShort.

A figura 6.1 mostra a tela inicial do BeShort, onde o usuário escolhe a conta que

[1] https://dev.twitter.com/
[2] http://twitter4j.org/en/index.jsp
[3] `https://twitter.com/BeShort2012_1`
[4] `https://twitter.com/BeShort2012_2`

deseja visualizar e manipular.


**Figura 6.1:** Tela inicial BeShort

A figura 6.2 mostra a *timeline* do usuário escolhido. A função da *timeline* é exibir todos os *tweets* que já foram postados pelo usuário, estas postagens seguem uma ordem cronológica do momento em que foi postada. O quadro número 1 é onde o usuário entra com o texto do *tweet* que é enviado através do botão "Envia *Tweet*" que está em destaque no quadro 2. Este comando de enviar faz com que URLs sejam identificadas no conteúdo do *tweet* e encurtadas antes de ser enviada para a rede social. O quadro 3 é responsável por acionar a função que mostra todos os *tweets* da conta do usuário e das contas dos usuários que ele segue.

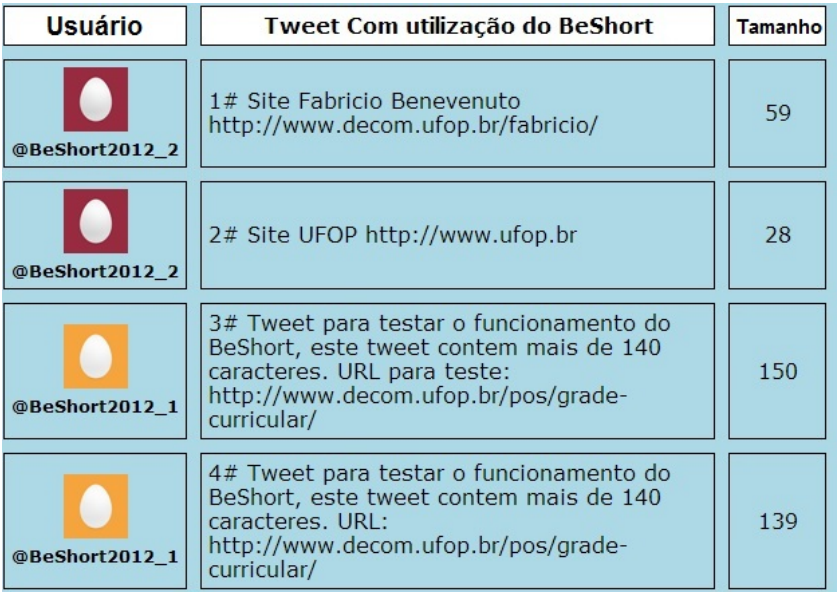
As figuras 6.3 e 6.4 mostram os *tweets* que já foram postados, sendo que a diferença entre as duas encontra-se no fato de usar ou não o BeShort para desencurtar às URLs. Na figura 6.3 o BeShort é utilizado, ou seja, antes de publicar a URL na *timeline* o BeShort é acionado para desencurtar a URL e fazer com que ela fique na sua forma original. Já na figura 6.4 o processo de desencurtar não acontece, assim a URL fica de maneira encurtada, impossibilitando seu uso.

Os exemplos mostrados ilustram o funcionamento do BeShort, neles podemos observar três pontos importantes. **1)** Existe a possibilidade de enviar *tweets* com mais que 140 caracteres. **2)** Às URLs não ficam ofuscadas ao serem exibidas para os usuários, como nos encurtadores atuais. **3)** Um ponto negativo é que usuários que não utilizam o


**Figura 6.2:** *Timeline* completa

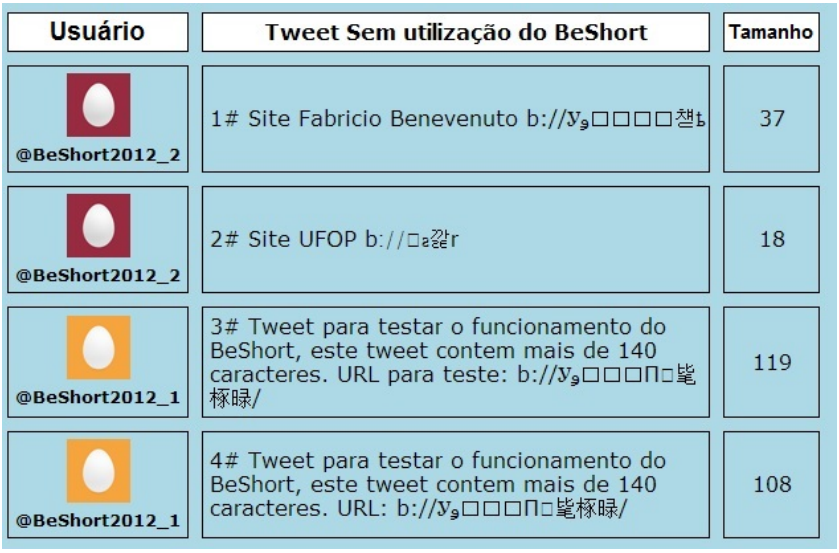
Beshort veriam a URL de forma encurtada e não poderiam desencurtá-la. Sendo assim, idealmente o Beshort deveria ser integrado como parte de um protocolo a ser adotado por todas as aplicações que utilizam a API do Twitter ou de outra rede social que queira adota-lo. Além disso, o Beshort poderia ser empregado por comunidades, organizações ou grupos fechados que pretendem trocar mensagens entre si.

Apesar dessa limitação, essa dissertação mostra que é possível realizar a compressão de forma descentralizada e sem perda de desempenho em relação aos sistemas encurtadores praticados atualmente. Além de conseguir taxas de encurtamento competitivas, o Beshort ainda possui outra grande vantagem em relação aos outros serviços é o fato de não ofuscar a URL original, concedendo uma maior segurança aos usuários de aplicações web. O funcionamento deste protótipo pode ser visto no sítio web `http://200.131.216.78:8080/bs`.



| Usuário | Tweet Com utilização do BeShort | Tamanho |
|---|---|---|
| @BeShort2012_2 | 1# Site Fabricio Benevenuto http://www.decom.ufop.br/fabricio/ | 59 |
| @BeShort2012_2 | 2# Site UFOP http://www.ufop.br | 28 |
| @BeShort2012_1 | 3# Tweet para testar o funcionamento do BeShort, este tweet contem mais de 140 caracteres. URL para teste: http://www.decom.ufop.br/pos/grade-curricular/ | 150 |
| @BeShort2012_1 | 4# Tweet para testar o funcionamento do BeShort, este tweet contem mais de 140 caracteres. URL: http://www.decom.ufop.br/pos/grade-curricular/ | 139 |

**Figura 6.3:** *Timeline* utilizando o BeShort para desencurtar



| Usuário | Tweet Sem utilização do BeShort | Tamanho |
|---|---|---|
| @BeShort2012_2 | 1# Site Fabricio Benevenuto b://y□□□□챌ㅏ | 37 |
| @BeShort2012_2 | 2# Site UFOP b://□ㅏ깔r | 18 |
| @BeShort2012_1 | 3# Tweet para testar o funcionamento do BeShort, este tweet contem mais de 140 caracteres. URL para teste: b://y□□□Π□뇥椓睩/ | 119 |
| @BeShort2012_1 | 4# Tweet para testar o funcionamento do BeShort, este tweet contem mais de 140 caracteres. URL: b://y□□□Π□뇥椓睩/ | 108 |

**Figura 6.4:** *Timeline* sem utilizar o BeShort para desencurtar

# Capítulo 7

# Conclusão e Trabalhos Futuros

O uso de mensagens curtas tem sido amplamente explorado em sistemas como o Facebook e Twitter. Parte desse encurtamento está associado à grande popularidade do uso de celulares e tablets para a postagem de mensagens, o que muitas vezes requer a redução da quantidade de texto a ser exibida aos usuários desses aparelhos. Sendo assim várias redes sociais impõem um limite superior no tamanho das mensagens (ex. no Twitter a mensagem é limitada a 140 caracteres), levando os usuários a utilizar um serviço encurtador de URLs para economizar espaço de suas mensagens. Os encurtadores normalmente reduzem uma URL com dezenas de caracteres para menos da metade do seu tamanho original. Apesar desses serviços serem muito úteis e conseguirem altas taxas de compressão, eles estão sendo utilizados como forma de esconder ataque maliciosos.*Spamers* e *phishers* estão ofuscando suas URLs através desses encurtadores. Outro ponto negativo é o atraso no redirecionamento, já que uma URL curta ao ser acionada faz o redirecionamento para a URL longa correspondente, esta ação faz com que o tempo de acesso fique maior comparado ao acesso direto a URL longa.

Neste trabalho analisamos a viabilidade de uma abordagem descentralizada para encurtar URLs. Esta abordagem executa um algoritmo de encurtamento de URL no momento do envio da mensagem à rede social que, ao ser recebida, é expandida para sua forma original e exibida aos usuários. Nossa abordagem é baseada na substituição de termos (partes da URL) frequentes encontradas em URLs por caracteres UTF-8, onde esse caracteres podem ser enviados através de APIs de redes sociais e normalmente não são encontrados em URLs.

Nossos resultados mostram que em relação as taxas de compressão nossa abordagem

é competitiva aos serviços praticados atualmente. Com o BeShort cerca de 80% das URLs analisadas obtiveram taxas de encurtamento superiores a 75%, já no Bit.ly cerca de 40% das URLs conseguiram a mesma marca. Comparando o BeShort ao TinyURL os resultados são ainda mais favoráveis ainda, 90% das URLs encurtadas com o BeShort obtiveram porcentagens de encurtamento superior a 73%, ao passo que apenas 25% dos encurtamentos do TinyURL conseguiram porcentagens acima desse valor. O BeShort se mostrou inferior para URLs com um número elevado de caracteres, mas foi observado em nossas análises que em média 18% das URLs possuem tamanho maior que 100 e apenas 7% excedem os 140 caracteres. Essas análises confirmam que nossa abordagem é viável como estratégia de encurtamento.

Além de proporcionar taxas de encurtamento competitivas, é importante ressaltar que o BeShort evita os problemas encontrados nos encurtadores tradicionais pois, o usuário visualiza e acessa a URL que foi postada originalmente. Com isso, evita que usuários maliciosos ofusquem suas URLs e que tenha algum atraso de redirecionamento, já que o acesso é feito direto à URL.

O BeShort pode ser implementado em diversos lugares, podendo ser nos próprios clientes através de *plugins* em navegadores, incorporado diretamente em aplicações da web, como redes sociais especificas (Twitter) e em serviços de emails.

Apesar de resolver os problemas dos encurtadores tradicionais o BeShort também tem seu ponto negativo. Usuários que não utilizam o BeShort e receberem uma URL encurtada por ele, irão vê-la na sua forma encurtada, e não haverá jeito de desencurta-la. Sendo assim, idealmente o Beshort deveria ser integrado como parte de um protocolo a ser adotado por todas as aplicações que utilizam a API do Twitter ou de outra rede social que queira adota-lo. Além disso, o Beshort poderia ser empregado por comunidades, organizações ou grupos fechados que pretendem trocar mensagens entre si.

Como prova de conceito do funcionamento do BeShort, foi construída uma ferramenta capaz de executá-lo. O código dessa ferramenta será disponibilizado para a comunidade científica de forma a difundir o BeShort e permitir comparações futuras. Além disso, a base de dados de URLs utilizada nesse trabalho, contendo 1 milhão de URLs encurtadas com o Bit.ly e com TinyURL juntamente de suas respectivas versões longas, será disponibilizada para permitir pesquisas futuras.

Como trabalhos futuros pretendemos investigar novas políticas para à seleção de termos, de maneira que possam gerar melhores resultados no encurtamento das URLs. Mudando para o contexto de segurança para os usuários, será estudada uma maneira de

incorporar um método de detecção de URLs maliciosas no próprio mecanismo de compressão do BeShort. Além disso, para uma melhor difusão das ideias dessa dissertação, pensamos em desenvolver plugins para navegadores (ex.: Firefox [1]), de maneira que o plugin irá atuar na identificação das URLs em aplicações web, e também irá realizar as funções de encurtar e desencurtar as URLS. Um outro importante aspecto que devemos observar é a forma de fazer atualizações no dicionário já que, o perfil das URLs certamente irá evoluir ao longo do tempo, e se tratando de um dicionário fixo ele pode ficar obsoleto com o tempo.

[1] http://www.mozilla.org/en-US/firefox/new/

# Referências Bibliográficas


[1] Demetris Antoniades, Iasonas Polakis, Georgios Kontaxis, Elias Athanasopoulos, Sotiris Ioannidis, Evangelos P. Markatos, and Thomas Karagiannis. we.b: The web of short urls. In *ACM Int'l conference on World Wide Web (WWW)*, pages 715–724, 2011.

[2] Fabrício Benevenuto, Jussara Almeida, and Altigran Silva. Explorando redes sociais online: Da coleta e análise de grandes bases de dados às aplicações. In *Mini-cursos do Simpósio Brasileiro de Redes de Computadores (SBRC)*, 2011.

[3] Fabrício Benevenuto, Tiago Rodrigues, Virgílio Almeida, Jussara Almeida, and Marcos Gonçalves. Detecting spammers and content promoters in online video social networks. In *Int'l ACM Conference on Research and Development in Information Retrieval (SIGIR)*, pages 620–627, 2009.

[4] Fabrício Benevenuto, Gabriel Magno, Tiago Rodrigues, and Virgílio Almeida. Detecting spammers on twitter. In *Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference (CEAS)*, 2010.

[5] Fabrício Benevenuto, Tiago Rodrigues, Adriano Veloso, Jussara Almeida, Marcos Gonçalves, and Virgílio Almeida. Practical detection of spammers and content promoters in online video sharing systems. *IEEE Transactions on Systems, Man and Cybernetics - Part B*, 2012.

[6] Yazan Boshmaf, Ildar Muslukhov, Konstantin Beznosov, and Matei Ripeanu. The socialbot network: when bots socialize for fame and money. In *27th Annual Computer Security Applications Conference*, New York, NY, USA, 2011.

[7] Meeyoung Cha, Hamed Haddadi, Fabrício Benevenuto, and Krishna P. Gummadi. Measuring User Influence in Twitter: The Million Follower Fallacy. In *Int'l AAAI Conference on Weblogs and Social Media (ICWSM)*, 2010.

[8] Sidharth Chhabra, Anupama Aggarwal, Fabrício Benevenuto, and Ponnurangam Kumaraguru. Phi.sh/$ocial: The phishing landscape through short urls. In *Annual Collaboration, Electronic messaging, Anti-Abuse and Spam Conference (CEAS)*, 2011.

[9] Zi Chu, Steven Gianvecchio, Haining Wang, and Sushil Jajodia. Who is tweeting on twitter: human, bot, or cyborg? In *26th Annual Computer Security Applications Conference*, 2010.

[10] comScore. comscore introduces mobile metrix 2.0, revealing that social media brands experience heavy engagement on smartphones. `http://www.comscore.com/Insights/Press_Releases/2012/5/Introducing_Mobile_Metrix_2_Insight_into_Mobile_Behavior`. Acessado em Novembro/2012.

[11] Helen Costa, Fabricio Benevenuto, and Luiz Merschmann. Detecting tip spam in location-based social networks. In *28th Annual ACM Symposium on Applied Computing*, 2013.

[12] Mark Davis. Moving to unicode 5.1. `http://googleblog.blogspot.com.br/2008/05/moving-to-unicode-51.html#!/2008/05/moving-to-unicode-51.html`. Acessado em Dezembro/2012, 2008.

[13] Olha Digital. Twitter gera meio bilhão de mensagens por dia. `http://olhardigital.uol.com.br/jovem/redes_sociais/noticias/twitter-gera-meio-bilhao-de-tuites-por-dia`. Acessado em Novembro/2012, 2012.

[14] Facebook. Facebook Press Room, Statistics. `http://www.facebook.com/press/info.php?statistics`. Acessado em Novembro/2012.

[15] Hongyu Gao, Jun Hu, Christo Wilson, Zhichun Li, Yan Chen, and Ben Y. Zhao. Detecting and characterizing social spam campaigns. In *ACM Int'l Conference on Internet Measurement (IMC)*, pages 35–47, 2010.

[16] Saptarshi Ghosh, Bimal Viswanath, Farshad Kooti, Naveen Kumar Sharma, Korlam Gautam, Fabricio Benevenuto, Niloy Ganguly, and Krishna Gummadi. Understanding and Combating Link Farming in the Twitter Social Network. In *21st International World Wide Web Conference (WWW'12)*, Lyon, France, 2012.

[17] Steven Gianvecchio, Zhenyu Wu, Mengjun Xie, and Haining Wang. Battle of botcraft: fighting bots in online games with human observational proofs. In *16th ACM conference on Computer and communications security*, 2009.

[18] Steven Gianvecchio, Mengjun Xie, Zhenyu Wu, and Haining Wang. Measurement and classification of humans and bots in internet chat. In *17th conference on Security symposium*, 2008.

[19] Chris Grier, Kurt Thomas, Vern Paxson, and Michael Zhang. @spam: the underground on 140 characters or less. In *17th ACM conference on Computer and communications security*, New York, NY, USA, 2010.

[20] Zoltán Gyöngyi, Hector Garcia-Molina, and Jan Pedersen. Combating web spam with trustrank. In *Thirtieth international conference on Very large data bases - Volume 30*, 2004.

[21] Amanda Lee Hughes and Leysia Palen. Twitter adoption and use in mass convergence and emergency events. In *2009 ISCRAM Conference*, 2009.

[22] Florian Klien and Markus Strohmaier. Short links under attack: Geographical analysis of spam in a url shortener network. In *23rd Conference on Hypertext and Social Media, HT' 2012*, Milwaukee, Wisconsin, USA, 2012.

[23] Kyumin Lee, James Caverlee, and Steve Webb. Uncovering social spammers: social honeypots + machine learning. In *33rd international ACM SIGIR conference on Research and development in information retrieval*, New York, NY, USA, 2010.

[24] Kyumin Lee, Brian David Eoff, and James Caverlee. Seven Months with the Devils: A Long-Term Study of Content Polluters on Twitter. In *AAAI Int'l Conference on Weblogs and Social Media (ICWSM)*, Jul 2011.

[25] Benjamin Markines, Ciro Cattuto, and Filippo Menczer. Social spam detection. In *Int'l Workshop on Adversarial Information Retrieval on the Web (AIRWeb)*, pages 41–48, 2009.

[26] Johnnatan Messias, Lucas Schmidt, Ricardo Rabelo, and Fabrício Benevenuto. Sigam-me os bons! transformando robôs em pessoas influentes no twitter. In *Brazilian Workshop on Social Network Analysis and Mining (BraSNAM)*, Curitiba, Brasil, 2012.

[27] Barack Obama. Utilizando twitter para campanha de 2012. http://twitter.com/BarackObama. Acessado em Novembro/2012.

[28] PRLOG. Just how many url shorteners are there anyway? http://www.prlog.org/10879994-just-how-many-url-shorteners-are-there-anyway.html. Acessado em Novembro/2012.

[29] Leena Rao. Twitter seeing 90 million tweets per day, 25 percent contain links. http://techcrunch.com/2010/09/14/twitter-seeing-90-million-tweets-per-day/. Acessado em Novembro/2012, 2010.

[30] Nielsen Online Report. Social networks & blogs now 4th most popular online activity. http://tinyurl.com/cfzjlt. Acessado em Março/2010, 2009.

[31] Tiago Rodrigues, Fabrício Benevenuto, Meeyoung Cha, Krishna P. Gummadi, and Virgílio Almeida. On word-of-mouth based discovery of the web. In *ACM SIGCOMM Internet Measurement Conference (IMC)*, pages 381–393, 2011.

[32] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Earthquake shakes twitter users: Real-time event detection by social sensors. In *In Nineteenth International WWW Conference. ACM*, 2010.

[33] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: analysis of a botnet takeover. In *16th ACM conference on Computer and communications security*, 2009.

[34] Jeannette Sutton, Leysia Palen, and Irina Shklovski. Backchannels on the front lines: Emergent uses of social media in the 2007 southern california wildfires. In *5th International ISCRAM Conference*, 2008.

[35] The Unicode Consortium, editor. *The Unicode Standard, Version 6.1 — Core Specification*. The Unicode Consortium, Mountain View, CA, 2012. http://www.unicode.org/versions/Unicode6.1.0/.

[36] Kurt Thomas, Chris Grier, Dawn Song, and Vern Paxson. Suspended accounts in retrospect: an analysis of twitter spam. In *ACM SIGCOMM conference on Internet measurement conference, IMC '11*, New York, NY, USA, 2011.

[37] Marisa Vasconcelos, Saulo Ricci, Jussara Almeida, Fabrício Benevenuto, and Virgílio Almeida. Tips, Dones and To-Dos: Uncovering User Profiles in FourSquare. In *ACM Int'l Conference on Web Search and Web Data Mining (WSDM)*, 2012.

[38] Nivio Ziviani. *Projeto de algoritmos: com implementações em Pascal e C*. Thompson, São Paulo, 2004.