УДК 004.02 DOi 10.30837/ bi.2022.1(98).13

Iryna Kyrychenko[1], Yehor Shamrai[2]

[1] Kharkiv National University of Radio Electronics, Kharkiv, Ukraine,
iryna.kyrychenko@nure.ua, ORCID: 0000-0002-7686-6439

[2] Kharkiv National University of Radio Electronics, Kharkiv, Ukraine,
yehor.shamrai@nure.ua

COMPARATIVE ANALYSIS OF URL SHORTENING ALGORITHMS: PERFORMANCE CONSIDERATIONS IN THE APPLICATION DOMAIN OF WEB SEARCH AND INFORMATION RETRIEVAL TECHNOLOGY

This article provides an in-depth overview and comparative analysis of URL shortening algorithms, with a focus on their performance and security. URL shortening is an important tool for creating concise versions of long URLs, particularly in platforms with character limitations. However, the increasing prevalence of cyber threats has raised concerns regarding the security of URL shortening. The paper examines a range of URL shortening algorithms, including Hash-based and Randomized algorithms, and compares their effectiveness based on factors such as URL length, randomness, and resistance to brute-force attacks. In addition, the article explores the security risks involved in URL shortening and suggests strategies for mitigating them. The paper also includes a performance analysis of different URL shortening algorithms, considering the time required to generate a shortened URL and redirect the user to the original URL. Finally, the article discusses the challenges of evaluating URL shortening algorithms and outlines potential avenues for future research in this area.

URL SHORTENING ALGORITHMS, PERFORMANCE EVALUATION, SECURITY CONSIDERATIONS, HASH-BASED ALGORITHMS, PERFORMANCE EVALUATION, RANDOMIZED ALGORITHMS

Кириченко І.В., Шамрай Є.О. Порівняльний аналіз алгоритмів скорочення URL-адрес: Міркування щодо продуктивності в прикладній області веб-пошуку та інформаційно-пошукових технологій. У цій статті представлено детальний огляд і порівняльний аналіз алгоритмів скорочення URL-адрес з акцентом на їхню продуктивність і безпеку. Скорочення URL-адрес є важливим інструментом для створення стислих версій довгих URL-адрес, особливо на платформах з обмеженнями на кількість символів. Однак, зростаюча поширеність кіберзагроз викликає занепокоєння щодо безпеки скорочення URL-адрес. У статті розглядається низка алгоритмів скорочення URL-адрес, включаючи алгоритми на основі хешування та рандомізовані алгоритми, і порівнюється їхня ефективність на основі таких факторів, як довжина URL-адреси, випадковість та стійкість до атак грубої сили. Крім того, в статті досліджуються ризики для безпеки, пов'язані зі скороченням URL-адрес, і пропонуються стратегії для їх зменшення. Стаття також містить аналіз продуктивності різних алгоритмів скорочення URL-адрес, враховуючи час, необхідний для генерації скороченої URL-адреси і перенаправлення користувача на оригінальну URL-адресу. Нарешті, в статті обговорюються проблеми оцінки алгоритмів скорочення URL-адрес і окреслюються потенційні шляхи для майбутніх досліджень у цій галузі.

АЛГОРИТМИ СКОРОЧЕННЯ URL, ОЦІНКА ПРОДУКТИВНОСТІ, МІРКУВАННЯ БЕЗПЕКИ,ХЕШ-АЛГОРИТМИ, ОЦІНКА ПРОДУКТИВНОСТІ, РАНДОМІЗОВАНІ АЛГОРИТМИ

Introduction

URL shortening is a process of converting a long URL to a shorter one, which redirects to the original URL. The shortened URL is used to share links, save characters in social media posts, or make it easier for users to remember a URL. URL shortening services have become popular due to their convenience, but they also have some disadvantages, such as security concerns and performance issues. In this article, we will review and compare various URL shortening algorithms based on their performance and security.

In today's digital age, URLs have become an essential component of our daily lives. They are used to access various resources on the internet, ranging from websites to applications. However, the length of URLs can sometimes be a hindrance, especially when it comes to sharing them on platforms that have character limits, such as Twitter. This is where URL shortening comes into play. URL shortening is the process of taking a long URL and creating a shorter, more manageable version that redirects to the original URL. This process has become increasingly popular over the years, with numerous URL shortening services available on the internet. However, with the rise of cyber threats, the security of URL shortening has become a major concern. To address this issue, researchers have developed various URL shortening algorithms that prioritize both performance and security. These algorithms aim to create short URLs that are resistant to cyber-attacks while ensuring fast and efficient redirection to the original URL

This is an overview and comparison of various URL shortening algorithms based on their performance. The paper presents a detailed analysis of the most commonly used URL shortening algorithms, including Hash-based, Bijective, and Randomized algorithms, among others. The analysis considers various factors, such as the length of the generated URLs, the randomness of the generated strings, and the resistance to brute-force attacks. Furthermore,

the paper discusses the security risks associated with URL shortening and the measures that can be taken to mitigate them. It also highlights the importance of using HTTPS [1] protocol for securing the redirection process.

Overall, this paper aims to provide a comprehensive overview of URL shortening algorithms and their associated security risks. It is intended for researchers and practitioners interested in improving the performance and security of URL shortening algorithms.

URL shortening is a process of converting a long URL to a shorter one, which redirects to the original URL. The shortened URL is used to share links, save characters in social media posts, or make it easier for users to remember a URL. URL shortening services have become popular due to their convenience, but they also have some disadvantages, such as security concerns and performance issues. In this article, we will review and compare various URL shortening algorithms based on their performance and security.

In addition to providing an overview and comparison of URL shortening algorithms, this paper also explores the various use cases of URL shortening, such as social media sharing, email marketing campaigns, and tracking user behavior. URL shortening is often used in social media sharing, where the character limit is restricted. Shortened URLs allow users to share links to websites or content that they find interesting or relevant without taking up too much space. In email marketing campaigns, shortened URLs are used to track clicks and user engagement. The ability to track user behavior provides valuable insights into the effectiveness of the campaign. However, despite the numerous benefits of URL shortening, it comes with its fair share of security risks. Attackers can use shortened URLs to launch phishing attacks, spread malware, and perform other malicious activities. Therefore, it is important to consider the security implications of URL shortening algorithms when developing and using them. Finally, the paper discusses the challenges associated with evaluating URL shortening algorithms and the future research directions in this field. It highlights the need for further research to improve the security and performance of URL shortening algorithms while considering the evolving threat landscape.

In summary, this paper provides a comprehensive overview of URL shortening algorithms [2], such as Base62 [3], their associated security risks, SEO [4] optimization and their performance comparison. It is a valuable resource for researchers, developers, and practitioners interested in improving the efficiency and security of URL shortening algorithms.

1. Problematic of URL shortening algorithms and main goals of shortening

URL shortening is a technique that has become increasingly popular in recent years, as it allows users to share long, complicated URLs in a compact and easy-to-read format. However, the practice of URL shortening also raises a number of concerns and challenges, which need to be addressed in order to ensure that users are able to use shortened URLs in a safe and effective manner.

URL shortening algorithms are used to convert lengthy URLs into shorter ones. This is done to make it easier to share links on social media platforms and other places where character limits are imposed. While URL shortening algorithms have become quite popular, they also come with several problems that need to be addressed.

Security concerns: One of the most significant problems with URL shortening algorithms is the security risks they pose. Because the original URL is obscured by the shortened URL, users may be redirected to malicious or harmful websites without their knowledge. Cybercriminals can use shortened URLs to disguise phishing scams, malware downloads, and other types of online threats. Additionally, shortened URLs can be vulnerable to hacking and tampering, which can compromise user data and privacy.

Link rot: Another problem associated with URL shortening algorithms is link rot, which occurs when the original URL is no longer valid or has been changed. Because shortened URLs rely on the original URL to redirect users to the correct destination, link rot can lead to broken links and frustration for users. While some URL shortening services provide automatic link checking and updating, this is not always the case.

Over-reliance on third-party services: Many URL shortening services are provided by third-party companies, which means that users may have limited control over the service and how it is used. If the service provider goes out of business or changes their policies, this can lead to disruptions in service and broken links. Additionally, third-party URL shortening services may collect user data or display ads, which can be a privacy concern for some users.

Duplication of URLs: Because URL shortening algorithms use a fixed-length code to generate the shortened URL, there is a risk of duplication. If two different URLs generate the same shortened code, this can lead to conflicts and broken links. While some URL shortening services use randomization or other techniques to minimize the risk of duplication, this is not always foolproof.

Dependence on service availability: Finally, URL shortening algorithms are dependent on the availability of the service provider. If the service experiences downtime or other disruptions, users may be unable to access the shortened URLs. Additionally, because URL shortening services are often free or low-cost, there is a risk of service providers going out of business or changing their policies without warning.

By the way, URL shortening comes with several security risks, such as phishing attacks, malware distribution, and the potential exposure of sensitive information. These

risks occur because the shortened URL may obscure the actual destination of the link, and users may not be aware of the actual website they are accessing.

Phishing attacks are a common security risk associated with URL [5] shortening. Attackers may use shortened URLs to redirect users to fake websites that look identical to legitimate ones, tricking users into giving away their sensitive information.

Malware distribution is another significant risk associated with URL shortening. Attackers can use shortened URLs to distribute malware, which can infect the user's device when they click on the link.

To mitigate these risks, several measures can be taken, such as:

1. HTTPS: Use HTTPS for all URLs, including shortened URLs, to ensure that all communications between the user's browser and the website are encrypted and secure.
2. URL scanners: Use URL scanners that can detect phishing [6] URLs and malware distribution URLs. These scanners can identify and block URLs that are known to be malicious.
3. User education: Educate users about the potential risks associated with shortened URLs and how to verify the destination of the link before clicking on it.
4. URL expiration: Implement URL expiration policies, which can prevent attackers from using shortened URLs for an extended period.
5. Randomized URLs: Use randomized URLs instead of sequential ones. This can make it harder for attackers to guess the destination of the URL.

By taking these measures, the risks associated with URL shortening can be mitigated, and the security of shortened URLs can be improved.

Overall, URL shortening is a useful tool for sharing links on the internet, but it is not without its challenges and limitations. In order to use URL shortening effectively, users need to be aware of the potential problems associated with this practice and take steps to address them, such as monitoring and updating their links regularly, using trusted URL shortening services, and being cautious when clicking on shortened links. By taking these steps, users can minimize the risks associated with URL shortening and enjoy the benefits of this powerful tool for sharing and communicating information online.

2. URL Shortening Algorithms

URL shortening algorithms are used to create shorter and more manageable versions of lengthy URLs. These algorithms take a long URL as input and generate a shorter URL that redirects to the original URL when clicked. URL shortening algorithms are widely used in various applications, including social media, email marketing, and other online platforms that have character limits. One of the main advantages of using URL shortening algorithms is that they can help save space and improve readability, particularly in social media platforms where character limits are strict. Shorter URLs also make it easier for users to remember and share links and can improve the overall user experience. Another advantage of URL shortening algorithms is that they can provide analytics and tracking capabilities, allowing users to track clicks and monitor the performance of their links. This can be particularly useful for marketing campaigns and other online promotions, as it allows users to track the effectiveness of their efforts and make informed decisions about future campaigns.

3. Hashing Algorithms

Hashing algorithm [7] converts a URL to a fixed-size hash value, which is then used as a short URL. The hash value is unique to each URL, and the algorithm ensures that it produces the same hash value for the same URL. Hashing algorithms are fast and efficient, but they are vulnerable to collisions. A collision occurs when two different URLs produce the same hash value, which can result in a security breach or incorrect redirection.

Hashing algorithms are a popular method used in URL shortening services to generate shortened URLs. Hashing is a mathematical function that takes an input value, such as a URL, and produces a fixed-length output value, known as a hash. Hashing is a one-way function, meaning that it is impossible to reverse the process and obtain the original input value from the hash.

When generating shortened URLs, the hashing algorithm takes the original URL and computes a hash value based on the contents of the URL. The hash value is then mapped to a shorter, more compact URL that redirects to the original URL when clicked. Because the hash value is unique to the original URL, it ensures that each shortened URL is unique and can be used to redirect users to the correct destination.

There are several popular hashing algorithms used in URL shortening services, including MD5, SHA-1, and SHA-256. MD5 is a widely used hashing algorithm that produces a 128-bit hash value, while SHA-256 [8]and SHA-1 are more secure hashing algorithms [9] that produce 160-bit and 256-bit hash values, respectively. The longer the hash value, the more secure the algorithm is, as it becomes more difficult for attackers to guess the original input value.

There are several advantages to using hashing algorithms in URL shortening:

1. Efficiency: Hashing algorithms are computationally efficient and can generate hashes quickly and easily. This makes them ideal for use in high-volume applications, such as social media and online advertising.
2. Uniqueness: Hashing algorithms produce unique hash values for each input value, which ensures

that each shortened URL is unique and can be used to redirect users to the correct destination.

3. Security: Hashing algorithms are secure [10], as they are one-way functions that make it difficult for attackers to reverse-engineer the original input value from the hash.

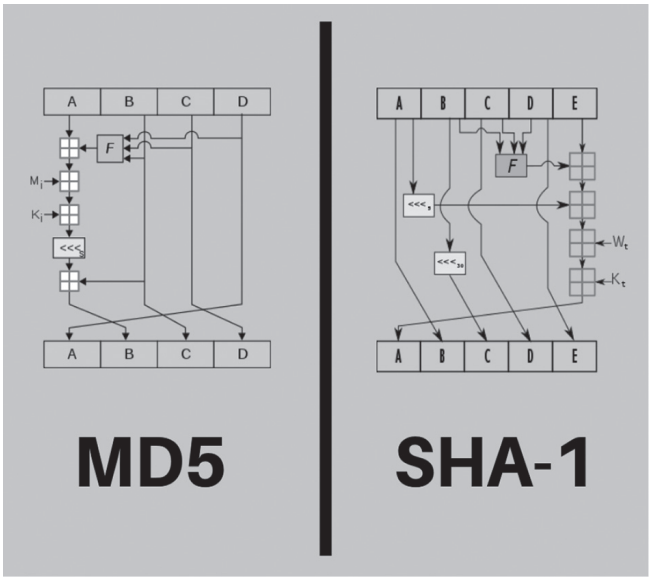

Fig. 1. Block of diagram for MD5 and SHA-1 processors

Random number generation and dictionary-based encoding are popular techniques used in combination with hashing algorithms for URL shortening. Random number generation involves generating a random number or string of characters and using it as a suffix or prefix to the original URL. This can help create unique and shorter URLs, as the length of the original URL no longer determines the length of the shortened URL. However, generating truly random numbers can be challenging, and collisions may still occur if the random numbers are not generated properly. Dictionary-based encoding, on the other hand, involves mapping the original URL to a compressed or encoded version using a predefined set of characters or symbols. This can result in shorter URLs compared to hash-based methods, and collisions are less likely as the encoding scheme is designed to avoid duplicate mappings. However, dictionary-based encoding can also be vulnerable to attacks, such as dictionary attacks or reverse lookups, which can compromise the security of the shortened URLs.

While hashing algorithms are a popular method for generating shortened URLs, they are not without their limitations. One limitation is that the length of the shortened URL is determined by the length of the hash value, which can be longer than other URL shortening methods. Additionally, hash-based URL shortening is vulnerable to collisions, where two different input values produce the same hash value, which can lead to duplicate URLs and other issues. To address these limitations, some URL shortening services use a combination of hashing and other techniques, such as random number generation or dictionary-based encoding, to create more secure and efficient shortened URLs.

4. Bijective algorithms

Bijective algorithms are commonly used for URL shortening because they are easy to implement, fast, and can create shorter URLs than other types of algorithms. A bijective function is a function that has a one-to-one correspondence between its domain and range, meaning that each input has a unique output, and each output has a unique input. This makes it ideal for generating unique, shortened URLs for long and unwieldy URLs. Functions can be injections (one-to-one functions), surjections (onto functions) or bijections (both one-to-one and onto). Informally, an injection has each output mapped to by at most one input, a surjection includes the entire possible range in the output, and a bijection has both conditions be true.

One of the advantages of bijective algorithms is that they do not require any additional storage space, as the shortened URL can be generated on-the-fly. This makes them particularly useful in situations where storage space is limited, such as in mobile applications or on websites with high traffic volumes.

Another advantage of bijective algorithms is that they can be easily customized to suit specific needs. For example, some applications may require shorter URLs than others, or may need to generate URLs with specific patterns or formats.

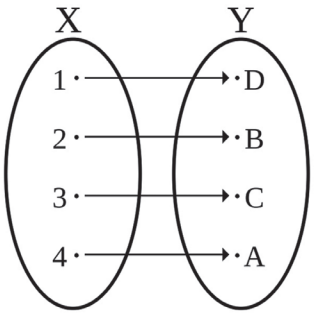

Fig. 2. Bijection between two sets

However, bijective algorithms do have some limitations. One of the main limitations is that the length of the shortened URL is directly proportional to the size of the domain space, which means that the shorter the URL, the smaller the domain space. This can make the algorithm more susceptible to collisions, where two or more URLs map to the same shortened URL. To mitigate this risk, many bijective algorithms use a combination of random numbers and hashes to generate unique, shorter URLs. This can increase the size of the domain space and reduce the likelihood of collisions.

5. Randomized algorithms

Randomized algorithms are algorithms that incorporate some element of randomness or probability into their operation. Instead of producing a deterministic output for a given input, randomized algorithms make use of a random number generator to generate some component of their output. This randomization can be used to improve the efficiency or accuracy of the algorithm, or to add an element of unpredictability or randomness to its behavior. One common application of randomized algorithms is in cryptography and computer security. For example, one of the most widely used cryptographic protocols, the RSA algorithm, relies heavily on random number generation to generate the keys used for encryption and decryption. Similarly, many secure communication protocols, such as SSL/TLS, incorporate randomization in order to prevent eavesdropping or interception of data.

There are several advantages to using randomized algorithms in security applications:

1. Randomness can make attacks more difficult: By incorporating randomization into the algorithm, it can be more difficult for an attacker to predict or analyze the behavior of the algorithm. This can make it more difficult for an attacker to find weaknesses or vulnerabilities that can be exploited.
2. Improved efficiency: Randomized algorithms can be faster or more efficient than deterministic algorithms, particularly for problems that are difficult to solve using traditional methods. This can be particularly useful for large-scale or computationally intensive security applications.
3. Unpredictability: Randomized algorithms can add an element of unpredictability or randomness to the behavior of a system. This can make it more difficult for an attacker to predict the behavior of the system, which can make it more difficult to launch successful attacks.

By the way, there are also some potential disadvantages to using randomized algorithms in security applications:

1. Potential for bias: If the random number generator used in the algorithm is biased or predictable, it can be exploited by an attacker to gain an advantage or to break the security of the system.
2. Difficulty of analysis: Randomized algorithms can be more difficult to analyze and prove correct than deterministic algorithms, particularly for complex systems. This can make it more difficult to identify or fix vulnerabilities in the system.
3. Complexity: Randomized algorithms can be more complex to implement and maintain than deterministic algorithms. This can make them more difficult to deploy or update, particularly in large-scale or distributed systems.

Randomized algorithms can be used for URL shortening by generating a random sequence of characters that represent the shortened URL. Here are some examples of randomized algorithms that can be used for URL shortening.

A simple approach is to generate a random string of characters to represent the shortened URL. This can be accomplished using a random number generator and a mapping of characters to their corresponding ASCII codes. The resulting string can then be appended to a base URL to create the shortened URL. Bloom filters are commonly used in URL shortening to check if a URL has already been shortened or not. A Bloom filter is a probabilistic data structure that uses multiple hash functions to map a URL to a set of bits in a bit array. When a new URL needs to be shortened, its presence in the Bloom filter is checked by computing the hash values using the same hash functions used to insert the URLs into the filter. If all the corresponding bits in the Bloom filter are set to 1, the URL is considered to be already shortened. If the URL is not already shortened, a randomized algorithm can be used to generate a unique short code for the URL. Bloom filters are widely used in URL shortening services because they are efficient in terms of memory and computation and can provide a high probability of detecting whether a URL has already been shortened. However, Bloom filters have a trade-off between memory usage and false positive rate. If the Bloom filter is too small or uses too few hash functions, the false positive rate can become unacceptably high, resulting in URLs being incorrectly flagged as already shortened. On the other hand, if the Bloom filter is too large or uses too many hash functions, the memory usage and computation time can become prohibitively high. Therefore, it is important to choose appropriate parameters for the Bloom filter based on the expected number of URLs and the desired false positive rate.

6. Comparison of URL shortening algorithms

We are going to cover experimental results of comparison of some type algorithms which can be used for URL shortening. As evaluation and comparison parameters for the algorithms, we will consider their performance and reliability. We will take one representative algorithm from each method of shortening URLs and conduct a detailed analysis, describing the essence of the algorithm and its implementation, testing its performance on different datasets, and comparing each of them. Many of those algorithms are used generally in neural networks and Face detection algorithms.

Let's take a look at SHA-2, Base62 and Bloom Filter algorithms for URL shortening.

6.1. SHA-1

SHA-2 is a family of cryptographic hash functions that were designed by the National Security Agency (NSA) and published by the National Institute of Standards and Technology (NIST) in 2001. The SHA-2 family includes six different hash functions, each with a different output

size: SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256. SHA-2 was created as a successor to SHA-1, which had been found to have vulnerabilities that made it insecure. The design of SHA-2 was based on the principles of the Merkle–Damgård construction, which is a popular method for building cryptographic hash functions. The SHA-2 family of hash functions has been widely used in a variety of applications, including digital signatures, password storage, and data integrity verification. It has also been used in conjunction with other cryptographic algorithms, such as RSA and AES, to provide secure communication over the internet.

The hash function is initialized with a set of constants that depend on the specific variant of SHA-2 being used. These constants are known as the initial hash values or initial states, and they are typically fixed for a given variant of the algorithm. For example, the initial hash values for SHA-256 are the first 32 bits of the fractional parts of the square roots of the first eight prime numbers. The input message is padded so that its length is a multiple of 512 bits. The padding is designed to ensure that the message is of a fixed length and to provide additional security against attacks. The padding is done in two steps:

1. Append a single '1' bit to the end of the message.
2. Append '0' bits until the length of the message is congruent to 448 modulo 512. This means that the message length will be a multiple of 512 after the padding, but there will be exactly 64 bits left to store the length of the original message.

The padded message is divided into 512-bit blocks, which are then processed by the hash function. Each block is processed in turn using a set of logical operations that depend on the specific variant of SHA-2 being used. The processing of each block consists of the following steps:

1. The 512-bit block is divided into 16 words of 32 bits each. These words are used to create a message schedule of 64 words, using a specific algorithm that depends on the specific variant of SHA-2 being used.
2. Initialize working variables: The working variables for the hash function are initialized using the initial hash values for the specific variant of SHA-2 being used.
3. Main loop: The message schedule and working variables are used in a series of logical operations to process the block. This loop is repeated 64 times for each block of data.
4. Update working variables: The working variables are updated based on the results of the main loop.

Once all of the blocks have been processed, the output of the hash function is generated. The output size depends on the specific variant of SHA-2 being used. For example, the output size for SHA-256 is 256 bits. The output is generated by concatenating the final hash values for each block, which are typically represented as a set of 32-bit words.

6.2 Base62

Method for encoding data in a format that uses a set of 62 alphanumeric characters, consisting of both uppercase and lowercase letters and digits. This algorithm is commonly used for shortening URLs and generating unique identifiers.

The Base62 algorithm uses a character set of 62 characters, consisting of 26 uppercase letters, 26 lowercase letters, and 10 digits. These characters are usually represented as a string of characters in the order in which they appear in the character set, such as "0123456789 ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklm nopqrstuvwxyz".

Before encoding the data using the Base62 algorithm, it must first be converted to a base-10 number. This is typically done using an existing encoding method, such as UTF-8 or ASCII. Once the data has been converted to base-10, it can be converted to base-62 using the following steps. Divide the base-10 number by 62, and record the remainder.Continue dividing the quotient by 62 and recording the remainders until the quotient is zero.

The resulting sequence of remainders represents the encoded data in base-62.

Each remainder in the sequence represents a numeric value that corresponds to a character in the Base62 character set. To encode the data, each remainder is mapped to its corresponding character in the character set.

The resulting sequence of characters represents the encoded data in the Base62 format. This encoded data can be used as a shortened URL or a unique identifier. The Base62 algorithm involves converting data to a base-10 number, dividing the base-10 number by 62 to obtain a sequence of remainders, mapping the remainders to characters in a character set, and outputting the resulting sequence of characters as the encoded data.

6.3 Bloom filter

The Bloom filter is a space-efficient probabilistic data structure that is useful for answering set membership queries. It was invented by Burton Howard Bloom in 1970. The main idea behind the Bloom filter is to use a bit array and a set of hash functions to determine whether an element is likely to be a member of a set or not. The filter has a fixed size, and elements are added to it by hashing them multiple times and setting the corresponding bits in the array. To query for membership, the same hash functions are applied to the element, and the corresponding bits in the array are checked. If all of the bits are set to 1, then the element is likely to be in the set. If any of the bits are set to 0, then the element is definitely not in the set.

The Bloom filter algorithm requires the definition of two parameters: the size of the filter and the number of hash functions. The filter size, denoted by m, is the number of bits in the filter, and the number of hash functions, denoted by k, determines the number of times the

input data will be hashed. The Bloom filter is initialized as a bit array of size *m* with all bits set to 0. The bit array is typically represented as an array of Boolean values. Alternatively, a bit vector or a bitmap can be used to represent the filter. To add an element to the Bloom filter, the element is hashed *k* times using *k* independent hash functions. Each hash function produces a hash value that is used to set the corresponding bits in the filter to 1. The specific bits to be set are determined by taking the modulo of the hash value with the size of the filter. In other words, the *k* hash functions produce *k* indices, and the corresponding bits in the filter are set to 1. To query whether an element is likely to be a member of the set, the element is hashed *k* times using the same hash functions used to add elements to the filter. The corresponding bits in the filter are checked, and if all of them are set to 1, the element is likely to be a member of the set. If any of the corresponding bits are 0, the element is definitely not a member of the set. One drawback of the Bloom filter algorithm is that it can produce false positives. This occurs when an element is not a member of the set but the corresponding bits in the filter are set to 1 due to collisions with other elements. The probability of a false positive depends on the size of the filter, the number of hash functions, and the number of elements in the set. To reduce the probability of false positives, the size of the filter and the number of hash functions must be carefully chosen based on the expected number of elements in the set. The Bloom filter algorithm is used in a variety of applications, such as web caching, network routers, distributed systems, and spell checkers. It is also used in DNA sequencing to filter out reads that do not match a reference genome. The Bloom filter is a simple and efficient data structure that can save memory and improve performance in many applications.

Table 1

Speed comparison of methods for different length of data

Technology	1 000	10 000	100 000
SHA-2	5 ms	8 ms	11 ms
Base62	2 ms	4 ms	5 ms
Bloom filter	3 ms	5 ms	9 ms

Base62 is an encoding scheme that converts binary data to a string using a character set of 62 characters (usually the 26 lowercase and 26 uppercase letters of the English alphabet, plus the 10 decimal digits). The resulting string is shorter than the original binary data, making it useful for shortening URLs or generating unique identifiers. Base62 encoding and decoding are simple operations that involve basic arithmetic and string manipulation, so they are very fast and have low computational overhead. Encoding and decoding a string using Base62 can be done in a few microseconds on modern CPUs.

SHA-2, on the other hand, is a family of cryptographic hash functions that generate a fixed-size digest of a given input data. SHA-2 is widely used for data integrity checking and digital signature verification, as well as password hashing and other security applications. SHA-2 is a computationally intensive operation that involves multiple rounds of bitwise operations and nonlinear functions, so it has higher computational overhead than Base62 encoding. The actual performance of SHA-2 depends on the size of the input data, the number of rounds used, and the hardware and software used to perform the computation. In general, SHA-2 can process data at a rate of several megabytes per second on modern CPUs and can take several milliseconds or more to compute a digest for a large input.

Bloom filters have a low memory overhead and can represent very large sets with a small amount of memory, but their accuracy depends on the size of the filter and the number of hash functions used. In terms of string shortening performance, Base62 is likely the fastest option, followed by Bloom filters and then SHA-256.For Base62 is a commonly used encoding scheme for shortening URLs, but it may not be optimal for hashing algorithms. Consider using a longer encoding scheme, such as Base64 or Base58, which provides more entropy and reduces the risk of collisions. Key stretching is a technique that makes brute-force attacks [11] more difficult by slowing down the hashing process. It involves iterative hashing the input with a salt and a fixed number of rounds, which can be adjusted to increase the computational cost of generating the hash. Ensure that the input to the hash function is properly validated to prevent attacks such as buffer overflows or injection attacks. Hash-based message authentication code (HMAC) can be used to add an additional layer of security to the hash function by incorporating a secret key into the calculation. This can prevent attacks such as message tampering or length extension attacks.

The choice of URL shortening algorithm depends on the specific requirements of the application domain. For applications that require speed and compactness, the Base62 encoding algorithm is a good choice. For applications that require security and uniqueness, the SHA-2 hashing algorithm is more suitable. Finally, the Bloom filter algorithm is a good choice for applications that require speed and scalability, but with lower accuracy requirements.

Conclusions

In conclusion, URL shortening algorithms play an important role in the domain of web applications, where shorter URLs can be beneficial for various reasons, such as improving usability, reducing the risk of errors, and tracking user behavior. However, the choice of algorithm is not trivial and requires careful consideration of the trade-offs between performance, security, and other requirements.

We discussed the challenges associated with URL shortening, including the need for uniqueness, collision

avoidance, and resilience to attacks. We also described the main goals of URL shortening, such as producing short, easy-to-read, and easy-to-share URLs that preserve the original destination. We then presented three categories of URL shortening algorithms, namely hashing algorithms [12], bijective algorithms, and randomized algorithms, and analyzed several specific examples, such as SHA-2, Base62, and Bloom filter. We compared the performance of these algorithms in terms of speed, memory usage, and collision rate for different lengths of data. By the way, all those algorithms can be used to solve other tasks in areas such as Big Data [13], Artificial Intelligence [14] etc. Our analysis revealed that each algorithm has its own strengths and weaknesses and is suitable for different use cases. For example, hashing algorithms are simple and fast but can suffer from collisions, while bijective algorithms are collision-free but may have limited scalability. Randomized algorithms can provide good trade-offs between speed, uniqueness, and security but require careful tuning.

Overall, our research showed that URL shortening algorithms are a complex and challenging area of study that requires a thorough understanding of the application domain and careful evaluation of the available options.

References

[1] *Beckett D.; Sezer S.,et al.* (2017). HTTP/2 Cannon: Experimental analysis on HTTP/1 and HTTP/2 request flood DDoS attacks, IEEE Xplore. doi: 10.1109/ICME.2007.4284716

[2] *Yuan, H.; Wun, B.; Crowley, P.; Beckett, et al.* (2010). Software-based implementations of updateable data structures for high-speed URL matching. IEEE Xplore.

[3] *Wen, S.; Dang, W.; Beckett, D.; Sezer, S.; et al.* (2018). Research on Base64 Encoding Algorithm and PHP Implementation. IEEE Xplore. doi: 10.1109/GEOINFORMATICS.2018.8557068.

[4] *Arias Aristizábal, L. F., & Duque Méndez, N. D.* (2012). SEO (Search Engine Optimization) schema application for websites with an emphasis on optimizing pages developed in flash. doi: 10.1109/ColombianCC.2012.6398011.

[5] *Soon L.-K., & Lee S. H.* (2008). Identifying Equivalent URLs Using URL Signatures, IEEE Xplore.doi: 10.1109/SITIS.2008.21.

[6] *Chen Y., Zhou Y., Dong Q., & Li Q.* (2020). A Malicious URL Detection Method Based on CNN. IEEE Xplore. doi: 10.1109/TOCS50858.2020.9339761

[7] *Yan, K., Chen, J., Cao, B., Zheng, Y., & Hong, T.* (2014). Research on a low conflict flow matching hash algorithm. IEEE Xplore. doi: 10.1049/cp.2013.2017.

[8] *Pham H. L., Tran T. H., Le V. T. D., Nakashima Y., & Yan K.* (2022). A Coarse Grained Reconfigurable Architecture for SHA-2 Acceleration: An Improved Low Conflict Flow Matching Hash Algorithm. IEEE Xplore. doi: 10.1109/IPDPSW55747.2022.00117

[9] *Kanca A. M.,* SAĞIROĞLU Ş. (2021). Sharing Cyber Threat Intelligence and Collaboration: An Improved Low Conflict Flow Matching Hash Algorithm. IEEE Xplore. doi: 10.1049/cp.2021.965432

[10] *Laatansa, R., Saputra, R., Noranita, B., Yan, K., Chen, J., Cao, B., Zheng, Y., & Hong, T.* (2020). Analysis of GPGPU-Based Brute-Force and Dictionary Attack on SHA-1 Password Hash: An Improved Low Conflict Flow Matching Hash Algorithm. IEEE Xplore. doi: 10.1049/cp.2020.8982390

[11] *Salamatian S., Huleihel W., Beirami A., Cohen, A., M dard M.* (2020). Centralized vs Decentralized Targeted Brute-Force Attacks: Guessing With Side-Information using an improved low conflict flow matching hash algorithm. IEEE Xplore. doi: 10.1109/TIFS.2020.2998949.

[12] *Studiawan, H., Pratomo, B. A., Anggoro, R., Yan, K., Chen, J., Cao, B., Zheng, Y., & Hong, T.* (2017). Clustering of SSH brute-force attack logs using k-clique percolation and an improved low conflict flow matching hash algorithm. IEEE Xplore. doi: 10.1109/ICTS.2016.7910269.

[13] *K. Smelyakov, A. Chupryna, O. Bohomolov and I. Ruban,* "The Neural Network Technologies Effectiveness for Face Detection," 2020 IEEE Third International Conference on Data Stream Mining & Processing (DSMP), 2020, pp. 201-205, doi: 10.1109/DSMP47368.2020.9204049.

[14] *K. Smelyakov, A. Chupryna, O. Bohomolov and N. Hunko,* "The Neural Network Models Effectiveness for Face Detection and Face Recognition," 2021 IEEE Open Conference of Electrical, Electronic and Information Sciences (eStream), 2021, pp. 1-7, doi: 10.1109/eStream53087.2021.9431476.

The article was delivered to editorial stuff on the 20.12.2022