2010

Delta bloom filter compression using stochastic learning-based weak estimation

Priyanka Trivedi
University of Windsor

Delta Bloom Filter Compression Using Stochastic Learning-Based Weak Estimation

By Priyanka Trivedi

A Thesis
Submitted to the Faculty of Graduate Studies
Through Computer Science
In Partial Fulfillment of the Requirements for
The Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

2010

Your file Votre référence
ISBN: 978-0-494-57595-6
Our file Notre référence
ISBN: 978-0-494-57595-6

AUTHOR'S DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.

ABSTRACT

Substantial research has been done, and sill continues, for reducing the bandwidth requirement and for reliable access to the data, stored and transmitted, in a space efficient manner. Bloom filters and their variants have achieved wide spread acceptability in various fields due to their ability to satisfy these requirements.

As this need has increased, especially, for the applications which require heavy use of the transmission bandwidth, distributed computing environment for the databases or the proxy servers, and even the applications which are sensitive to the access to the information with frequent modifications, this thesis proposes a solution in the form of compressed delta Bloom filter.

This thesis proposes delta Bloom filter compression, using stochastic learning-based weak estimation and prediction with partial matching to achieve the goal of lossless compression with high compression gain for reducing the large data transferred frequently.

Keywords: Bloom filter, Compression, Stochastic learning-based weak estimation, Delta Bloom filter, PPM.

DEDICATION

This thesis is dedicated to

My Family

ACKNOWLEDGEMENTS

This thesis is a result of the research conducted at the University of Windsor. In the process, some people who contributed in varied ways to the research and the creation of the thesis deserve special mention.

I would like to convey my sincere gratitude to my supervisors Dr. Xiaobu Yuan and Dr. Luis Rueda, for their excellent supervision, advice and guidance through out this research work. Without their help and guidance the work presented here would not have been possible. Their confidence in my abilities has been unwavering, and has helped to make this thesis a solid work. Above all and the most needed, they provided me unflinching encouragement and support in various ways. I hope to keep up our collaboration in the future.

I would like to thank my internal reader, Dr. Alioune Ngom and my external reader, Dr. S. Ejaz Ahmed for their participation as my thesis committee members and for taking out their precious time to review this thesis and to give their comments and suggestions for the thesis work. I would also take the opportunity to thank Dr. Akshai Kumar Aggarwal for being the Chair of Defense.

I gratefully acknowledge Dr. James Frank for his advice and contribution in bringing the thesis to this level and Dr. Maria Blass for her support at the time when I needed it the most. I would like to thank my family for their constant support in every possible way.

It is my pleasure to thank all those who made this thesis possible.

TABLE OF CONTENTS

LIST OF THE TABLES

LIST OF THE FIGURES

Chapter 1
INTRODUCTION

Computers and the technology related with computers have evolved over time. In seventies, memory requirements were a major issue. The reason for it was that the memory available at that time was small and expensive. Therefore, for commercial and research purposes it was essential to utilize memory resources as efficiently as possible.

In recent years, the scenario has evolved, though the core issues remain the same. The data to be dealt with has increased exponentially. Although, memory capacities have increased since the last decades, with the present data requirements, there is a substantial need to handle the data sets as concisely as possible. On the network side, also, there is a limitation on the bandwidth and the communication times and costs, since the data residing at any one node itself are significantly large. Here, also, there is a need for a concise data representation. Membership query is the most common query type and extensively used. It would reduce the relative costs sufficiently, even if there was just a way to know in advance and succinctly about the membership of an element, i.e. whether the element exist in the set or not.

Bloom filters have the capacity to fulfill all these requirements (broadly speaking, *three aspects of Information Economy*), though, with a small limitation. Therefore, there was a boom in the research work related to Bloom filters and their active participation in the recent applications. In fact, its use has been picked up for the web applications which are in demand in the industry.

Bloom filters are now widely used in peer-to-peer network systems [YS2006] and the research on them is still active in this field. The applications are for string matching, confidentiality maintenance, or the security [LPKS2005]. Bloom filters are also used for wireless podcasting and the association rules in data mining [QLW2007]. Bloom filters have found place in the forensic science to efficiently aggregate and search hashing

information. md5bloom [RCBG2006], a Bloom filter manipulation tool that can be incorporated into forensics, has been introduced. Recently Microsoft produced a Bloom filter forwarding architecture called *BUFFALO* which uses a small SRAM to store one Bloom filter of the addresses associated with each outgoing link. BUFFALO significantly reduces memory cost, increases the scalability of the data plane, and improves packet-forwarding performance [MS2009]. A data popularity conscious Bloom filter has been introduced recently [ZLSS2008]. It considers object popularity in sets and membership queries and minimizes false-positive probability of BF by adapting the number of hash functions used for each data object to its popularity in sets and membership queries. An aging BF with two BF for the dynamic set is introduced to remove stale data in the BF to make space for the new data. This scheme utilizes memory space more efficiently than double buffering, the current state of the art [Yoon2010].

As technology is evolving further, now, there is a need for improving the efficiency and representation of compressed Bloom filters. The applications involving heavy use of the bandwidth, distributed computing environment for databases or proxy servers, and even applications that are sensitive to the access to the data with frequent modifications, all will benefit from the use of compressed Bloom filters. Depending on the benefits that compressed Bloom filters offer, they will prove of vital use in application areas such as databases, web caching, P2P applications and network applications.

Moving ahead of this, applications, for example in web proxies, that broadcasts its cache content or the updates to the other proxies at frequent intervals, will benefit more from the compressed delta Boom filter, which represents the updates to the Bloom filter in the compressed form. In this way less data has to be transmitted.

In this thesis, delta Boom filter compression is proposed using Stochastic learning-based weak estimation (SLWE) [OR2005] and using prediction with partial matching (PPM). For delta Bloom filter compression, a lossless compression technique is needed to reduce the amount of data without any loss of data. Using SLWE, probabilities of the source symbols can be adaptively updated while being encoded. This technique not only uses

statistics of the source but also considers the variability of the source statistics. PPM is a higher-order source modeling technique that makes use of the past history of the data being encoded to give more efficient compression.

The proposed methods will provide substantial compression gain to applications relying on heavy use of the bandwidth and distributed computing environment relying on frequent updates of the underlying data.

1.1. Motivation

Bloom filters have a widespread use in databases, web caching, P2P applications and network applications both for data storage and as a message. There is a present need for an efficient and compact transfer of Bloom filter updates. This thesis addresses this need and proposes delta Bloom filter compression using SLWE and PPM based methods.

Why Bloom Filters?

A Bloom filter is a probabilistic data structure which is used to test whether an element is a member of a set, i.e. membership testing. Given a set $S = \{x_1, x_2, x_3, \ldots, x_n\}$ on a universe U, if there is a need to answer queries of the form: Is $y \in S$?, a Bloom filter is a good choice. Bloom filters provide an answer in a constant time depending on the time to hash. It requires a small amount of space compared to the original data.

Why Delta Bloom filters?

A Bloom filter is not just a data structure; it can also be used as a message which needs to be broadcasted. In applications, for example, which require web cache sharing, the proxies periodically broadcast updates of their cache contents. It has been proved that Bloom filter-based caches are more efficient than the exact dictionary or the server list approaches [FCAB2000]. In this scenario, where broadcasting of the updates is required, a delta Boom filter can be sent instead. Applications that rely on the access to the data,

which is frequently updated, can gain from the use of the delta Boom filter. Furthermore, this delta Boom filter can be compressed to increase the compression gain for these applications and this is the main aim that we endeavor in this thesis.

Why Compression?

In a situation in which the Bloom filter is not just a data structure but also a message and this message needs to be transmitted from one location to another, it will be beneficial to compress it. Where there is a need for the access to this data which is frequently updated and transmitted, compressed delta Boom filter can prove beneficial. Compression will reduce the amount of the data being transmitted.

Why Lossless Compression?

Lossless compression is a process in which the data recovered after decompression is exactly same as the original data. Bloom filters represent the set in the form of a bit vector. Each element is hashed k times $h_1,...h_k$ with the range $\{1,....m\}$, i.e. $h_i : X \rightarrow \{1....m\}$, $1 \leq i \leq k$. To store an element $X \in S$, the bits $h_i(X)$ are set to 1 for $1 \leq i \leq k$. If any of these bits are incorrectly represented due to the loss of data during compression, the element which is a member of the set might be incorrectly represented as not a member. This defies the basic property of the Bloom filter which states that the false negative rate is not a characteristic of a Bloom filter [Mitz2002]. Therefore, lossless data compression is the only choice for Bloom filter Compression.

Why SLWE based Compression?

Stochastic Learning-based Weak Estimation (SLWE) is a fairly new technique introduced in [OR2006]. To the best of our knowledge, it has not been used to compress Bloom filters. Using this technique we can adaptively update the probabilities of the source symbols while being encoded. This technique not only uses higher-order statistical models about the source but also considers the variability of the source statistics. Using

this technique, the data is encoded through single pass unlike the static coding algorithms which require two passes, one pass to learn the probabilities and the second to encode the data. As opposed to higher models, SLWE based compression has linear space complexity. Also, in our experiments it was found out that SLWE achieves a better compression gain than benchmark arithmetic coding and the higher–order Prediction with partial matching compression method. Our proposed delta Boom filter compression method using SLWE also works better in terms of scalability and adaptability in comparison to the above mentioned methods. This thesis also proposes delta BF compression using PPM model with arithmetic coding.

1.2. Thesis Contribution

This thesis proposes delta Boom filter compression. With respect to this goal, the thesis contribution is twofold. Firstly, the use of higher-order model with arithmetic coding, in terms of prediction with partial matching, for achieving delta Bloom filter compression. Secondly, the use of SLWE an advanced estimation technique with arithmetic coding that uses higher-order statistical model and, also, takes into account the variability of the source statistics, for compressing the delta Bloom filter.

The *compression gain* achieved by the proposed methods is compared with the benchmark arithmetic coding with adaptive unigram model with a conclusion that the proposed methods are more efficient. Also, the results are compared between the proposed delta Bloom filter compression using higher-order prediction with partial matching and arithmetic coding, and found that the second proposed method for delta Boom filter compression using SLWE is better in terms of compression gain.

The proposed methods are tested for the efficiency in terms of the compression gain, scalability in terms of the dependence on the size of the Bloom filter and the adaptability in terms of the number of the hash functions used, delta change in the Bloom filter content, and dependence on the parameter λ (in case of SLWE).

1.3. Thesis Summary

The thesis is organized as follows:

Chapter 2: Bloom filter: In this chapter, Bloom filter principles are explained with its characteristics and limitations. We highlight why Bloom filters are widely accepted for the applications. Few uses of Bloom filters for networking and web applications are also discussed. This cements the fact that, although Bloom filters were introduced in 1970's [Bloom1970], they have an active participation in the computer field. In this chapter, the principles of the delta Boom filters are also explained.

Chapter 3: Data Compression: This chapter discusses data compression in the past and in the present scenarios and how it is the behind-the-scene enabler for the common aspects of our life. This chapter presents different data compression techniques including lossless compression. Then, the encoding methods and the higher-order models for coding are discussed. This chapter explains Huffman and arithmetic coding techniques, as well as higher-order models, including Prediction with partial matching, Burrows-Wheeler transformation and most importantly, the Stochastic learning- based weak estimation method which is the backbone of the thesis, are also presented here.

Chapter 4: Problem Specification and Solution: This chapter presents the delta Boom filter in detail including the real world applications and importance of the proposed compressed delta Boom filter and how it can play an active role in enhancing the efficiency of the present model. First, the implementation of the proposed method for delta Boom filter compression using a higher-order model in terms of prediction with partial matching coupled with arithmetic coding is presented. Then, the implementation details of the second proposed method for delta BF compression by using the Stochastic learning-based weak estimation method are presented.

Chapter 5: Implementation and Experiments: This chapter presents the issues faced during the implementation and the experimental setup. Then the results of the proposed delta Boom filter compression using the Stochastic learning-based weak estimation method are presented. The proposed method is tested for efficiency in terms of compression gain, scalability in terms of the dependence on the size of the Bloom filter and adaptability in terms of the number of the hash functions used, delta change in the Bloom filter content and its dependence on the parameter λ.

Then the efficiency and dependence of the proposed methods are compared to the benchmark and among each other on the factors mentioned above. With the encouraging results of the comparison, it can be stated that the second proposed method that utilizes SLWE is better than the benchmark arithmetic coding with adaptive unigram model and the higher-order model that utilizes prediction with partial matching.

Chapter 6: Conclusion and Future Work: This chapter concludes the thesis by highlighting the contribution and the possible avenues for future work.

Chapter 2
BLOOM FILTERS

This chapter presents Bloom filters. First, the concept of probabilistic data structure is presented and then the Bloom filter, a type of probabilistic data structure is introduced with its characteristics. Then, Bloom filter applications are discussed. At the end, the principle of the delta Boom filter is presented.

2.1. Probabilistic Data Structure

A *probabilistic data structure* (PDS) is a data structure that supports algorithms which inherently use a certain amount of randomness for their operations. Without a measure of randomness, the data these data structures encode would use a prohibitive amount of memory and/or the algorithms they support would be very slow. As a trade off for their space, speed, and computational efficiency, many PDSs allow for a small margin of error returns for instance Bloom filters.

2.2. Standard Bloom Filter

The Bloom filter was first introduced by Burton H. Bloom in 1970 [BLOOM1970]. It is a probabilistic data structure that is used basically to test whether an element is a member of a set, i.e. membership testing. Here, false positives are possible, but false negatives are not. Elements can be inserted into the set, but not removed/deleted. There are Bloom filter variants that have achieved this (counting Bloom filter). The application areas which require error free performance are not meant to use Bloom filters.

A Bloom filter represents an n-element set $S = \{X_1, X_2, ..., X_n\}$ by using a bit vector $V = V_1, V_2, ... V_m$ of length m. All the bits are initially set to zero as in Figure 2-1 (a). The filter uses k independent hash functions $h_1, ...h_k$ with the range $\{1,....m\}$, i.e. $h_i : X \rightarrow \{1....m\}$, $1 \leq i \leq k$.

As in Figure 2-1 (b), to store an element $Q \in S$, the bits $h_i(Q)$ are set to 1 for $1 \leq i \leq k$. It is to be noted that although a location can be set to one multiple times, it is only the first change that has an effect. With respect to Figure 2-1 (c), if we want to check whether the element Q is in the set S, we simply check if all $h_i(Q)$ are set to 1 or not. If any is set to 0, then Q is not a member of S.

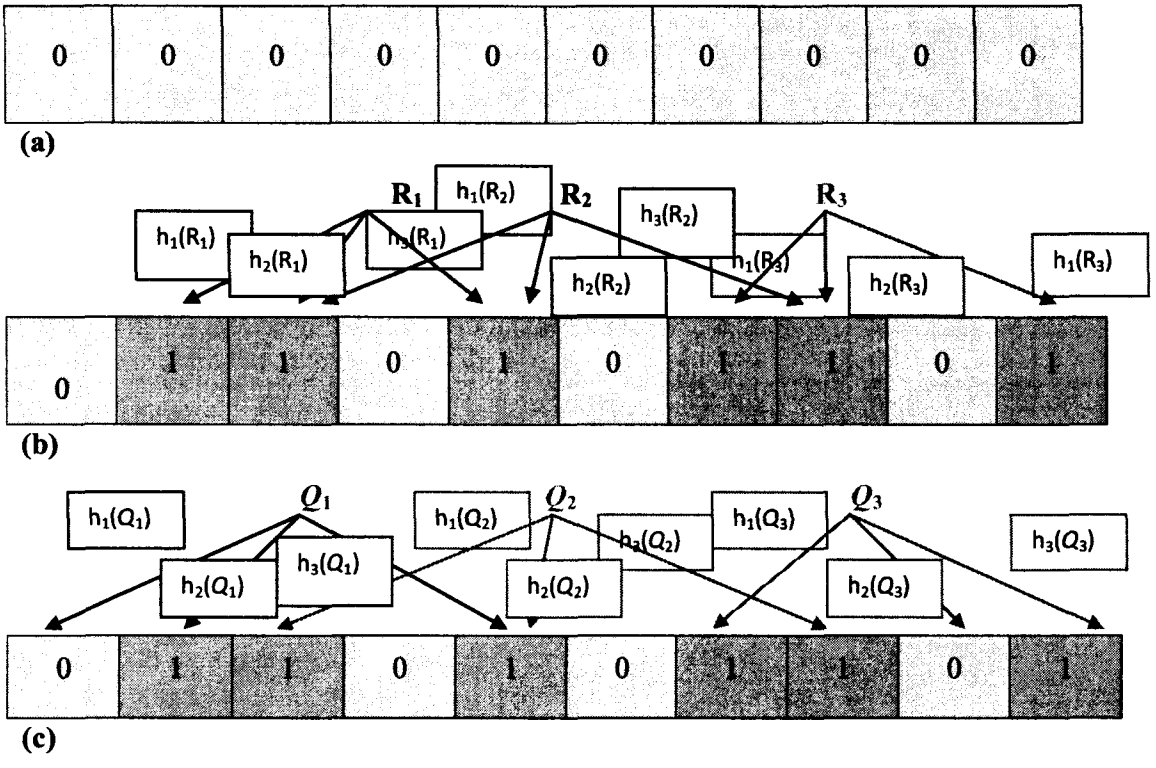

Figure 2-1: Standard Bloom filter

(a): all the bits are initially set to zero.

(b): each element is hashed k times and corresponding bits are set to one.

(c): While membership testing, Q_1 and Q_3 cannot be members of the groups as they have a 0 value in the bit corresponding to them.

2.2.1. False Positive Rate / False Drop Rate

If all h_i (Q) are set to 1, we should not infer from it that the element Q is definitely in S. There is a reason behind this. It is also possible that by coincidence $h_1(Q),\ldots,h_k$ (Q) are all set to 1. In the above example in Figure 2-1(c), $h_1(Q_2)$, $h_2(Q_2)$ and $h_3(Q_2)$ all bits are set to 1, yet there is a probability that Q_2 is not the element of the set. This is known as the *false–positive/drop.* The probability of this happening is termed as the *false-positive rate*. Here, it is to be noted that the Bloom filter does not have a false-negative but may have false-positive. This means that if an element does not exist in the set, the answer could be that it does exist. On the other hand, if the element does exist in the set, it is never possible that it is reported not to exist. In Figure 2-1(c), Q_1 and Q_2 cannot be in the set as there is a 0 value corresponding to them.

When all the elements are hashed into the Bloom filter, the probability that a specific bit is still 0 is:

$$(1-1/m)^{kn} \approx e^{-kn/m}$$

Suppose that $p = e^{-kn/m}$. Then, the probability of a false positive, on the assumption that the hash functions are purely random, is:

$$f = \left(1-e^{-kn/m}\right)^k = (1-p)^k \text{ , where } p = e^{-kn/m}$$

The false-positive rate depends on the total length of the filter and the number of elements that it contains. In other words, it means that the smaller the size of the Bloom filter and larger the number of elements it contains, the greater the false positive rate is. It is to be noted that if the probability, i.e. the false positive rate, is small enough, the false positive is accepted for many applications. This depends on the advantages of using the Bloom filter outweighing the false positive disadvantage.

The overhead borne due to the addition of every hash function remains constant, whereas, after a given threshold value the compounded benefit of adding this new hash function starts decreasing. The probability of false positives can be reduced by the allocation of a larger amount of memory [Mitz2002].

2.3. Bloom Filter Applications

Bloom filters have a wide spread use for the applications that require data for membership testing or similarity detection. These applications can rely heavily on the bandwidth. In this section, the Bloom filter use for network and web based applications is presented. It is relevant to discuss these applications as these are the fields which can also benefit in the future from the compressed delta Boom filters.

2.3.1. Using Bloom filters for Network Applications

Bloom filters upon their introduction, were not considered for their use in network applications. As the data residing at any one node itself grew exponentially, it was realized that there is a limitation on the bandwidth, communication time and cost. The need for a concise data structure like the Bloom filter was felt and, consequently, they were implemented into various areas within networking as listed in Figure 2-2. The research on the applications in networking is later summarized in Table 2-1.

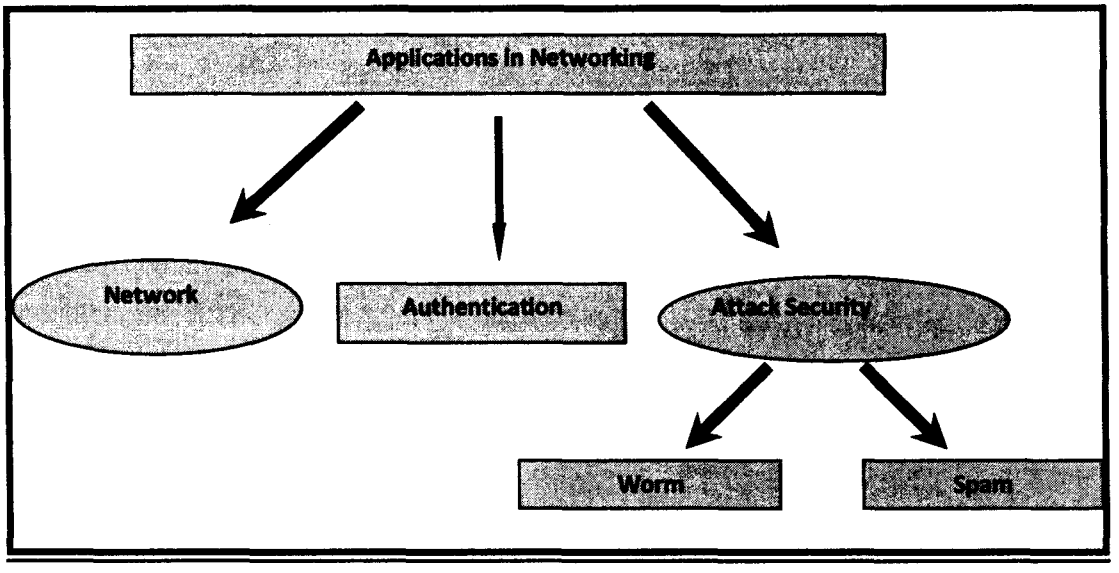

Figure 2-2: Applications in Networking

2.3.1.1. Broadcast Authentication

As broadcast authentication is the most basic approach to defend against attackers in the wireless environments, it is a critical security primitive for sensor networks. The central server relies on it to issue legitimate commands or queries to all or partial motes in the network. μTESLA is a light-weight and self-healing protocol for sensor motes. Chen *et al* [CLL2008] have proposed an efficient and flexible broadcast authentication protocol combing μTESLA and compressed Bloom filter techniques. This technique also supports multiuser authentication.

To bundle W independent μTESLA instances, Curtain takes the last N keys as entries to be hashed into the Bloom filter of m bits. The Bloom filter bitmap replaces the last-generated key in standard μTESLA or the root key in tree-based μTESLA as the commitment. Pre-loaded by the sender to each receiver, the bitmap provides the computational security during the usage of W μTESLA instances. For the receiver to activate the next instance, it just checks that the last-generated N keys are all successfully hashed into the Bloom bitmap. Before the end of the W instances, the sender broadcasts the compressed bitmap of the next W instance with an ECC signature to enter the next session.

The hash functions used are based on SHA1 and a variant of first-order difference compression is used to compress the bitmap [CLL2008]

2.3.1.2. Network Routing

Longest Prefix Matching (LPM) techniques are important for networking due to their fundamental role in the performance of the Internet routers. Internet routers are required to search variable-length address prefixes for the Classless inter-domain routing (CIDR) in order to find the longest matching prefix of the IP destination address and retrieve the corresponding forwarding information for each packet traversing the router. This is often the performance reducer for the high-performance internet routers.

A probable solution given by Dharmapurikar *et al* [DKT2003] is sorting the forwarding table entries by prefix length, associating a Bloom filter with each unique prefix length, and "programming" each Bloom filter with prefixes of its associated length. A search begins by performing parallel membership queries to the Bloom filters by using the appropriate segments of the input IP address. The result of this step is a vector of matching prefix lengths, some of which may be false matches. Hash tables corresponding to each prefix length are probed in order of longest to shortest match in the vector, terminating when a match is found or all of the lengths represented in the vector are searched. The best part is that the approach is feasible for the newer version Internet Protocol Version 6 (IPv6) which uses four times larger destination address of 128 bits.

2.3.1.3. Wireless Networks

A wireless object network is a network that connects objects to each other wirelessly. Due to hardware advances it will be possible to embed small devices into everyday objects such as toasters and coffee mugs, and hence forming a network. Wang *et al.* presented a search engine for such a network called Snoogle. Snoogle uses information retrieval techniques to index information and process user queries. It uses Bloom filters to

reduce communication overhead. Snoogle also considers security and privacy protections for sensitive data [WTL2008].

2.3.1.4. Authentication

There are two major components for entity authentication for pervasive computing environments: the Master key and the multiple access tokens. Master keys are convenient, whereas multiple access tokens improve usability and do not have the revocation problems. The question is how to combine both the advantages of traditional master keys and multiple access tokens while avoiding their disadvantages. Zhu *et al.* [FML2006] present the Master Key method, which is an approach for digital access tokens to have the advantages of master keys and multiple access tokens.

The Master Key is invoked when the owner pushes a lock (target resource) or unlock button. At that time, the Master Key discovers the right key (digital access token) for a lock and authenticates with the lock. Now, the master key has to deal with privacy, security and scalability- efficiency problems. The Bloom filter approach is useful here. The Master key can utilize Bloom filter's time and space advantages as well as false positive cases. The Bloom filter approach also helps in maintaining the security as cryptographic hash functions, such as MD5 and SHA, are used.

Code words in the master key approach have two formats: the hash format and the Bloom filter format. The hash format is used when the key and lock has a one-to-one relation, for example, after the Master Key has discovered the target lock. The Bloom filter format is used when keys and locks have one-to-many relationships. For instance, the Master Key queries a set of potential locks, or a lock needs to identify a particular key owner from many key owners.

However, here, the Master Key does not support multiple groups of key owners. There are Bloom filter variants which support multisets. These could be investigated for the future work. Moreover, compressed Bloom filter can be used for further improvement.

2.3.1.3. Network Security

2.3.1.3.1. Worm Attack

It is a fact now that the manual defense techniques are not useful against the worm epidemic. Due to the high infection speed of worms, worm identification and containment methods are deployed into the network (instead of end-hosts) for fast and coordinated response with a high coverage. Signature-based filtering has a higher chance of containing epidemics and precision compared to blacklisting methods.

The fundamental operation of signature-based filtering is deep packet inspection (DPI), the string matching of packet payloads against characteristic worm content, known as signatures [AC2005]. Artan *et al* presented a space-efficient method to follow and detect signatures that are fragmented over multiple packets, to solve the multi-packet signature detection problem. To solve this problem, they introduced a new data structure, called prefix Bloom filters (PBFs) and Chain heuristic (CH).

PBFs help recognize prefixes of signatures so that signatures over multiple packets can be detected. These are Bloom filters programmed with the prefixes of the signatures. P_i is taken as the set of i-length prefixes of all signatures with lengths longer than i and S_i is the set of signatures with lengths i. Then, PBF_i is a Bloom filter that is programmed with the elements of the set P_i and SBF_i is a Bloom filter programmed with the elements of set_i.

The major advantage of the Prefix Bloom filter is that, even if a string contains only a prefix of the signature, this prefix can be detected due to the use of the PBF. Also, to make sure that the prefixes are detected just like the signatures, the Bloom filter is used to store these signature prefixes. However, this increases the load on memory requirements. Therefore, another concept called Chain heuristic is introduced to reduce this increased memory requirement. This Chain heuristic is used to decrease the false positive rate of

prefix Bloom filters as well as the SBFs, without the requirement of any additional memory.

2.3.1.3.2. Spam Filters

Spamming is the abuse of electronic messaging systems to indiscriminately send unsolicited bulk messages. The filters are used as a safeguard against the spam by classifying email as spam or ham (non-spam) by a statistical analysis of the message's header and content (body). Statistical-based Bayesian filters are widely used in the tools for blocking spam.

However, they have a problem of being bounded by the memory access rate. Kang. *et al.* [KZ2006] have used Bloom filters for speeding up spam filters while keeping high classification accuracy. First, the method used approximates of the dictionary lookup with hash-based Bloom filter lookup, which trades off memory accesses with increase in computations. Second, the method uses lossy encoding.

The approach takes Bloom filters one step ahead and uses a Bloom filter to serve for value queries while preserving the Bloom filter's desired operating characteristics. For a given token in the member set, the extension returns a value that corresponds to a given token as the spam filters must retrieve each token's associated probability value.

Table 2-1: Summary of the Research on the use of Bloom filters for Network Applications.

Author(s)	Conference/Journal and Year	Title of the Paper	Main Contribution
Wang, H., Tan, C.C., Li, Q	IEEE *INFOCOM 2008*	Snoogle: A Search Engine for the Physical World	The authors propose a search engine, Snoogle, for wireless object networks. It uses a BF to reduce communication overhead.
Chen, Y., Lin, I., Lei, C., and Liao, Y.	Lecture Notes in Computer Science, 2008	Broadcast Authentication in Sensor Networks Using Compressed Bloom filters	The authors propose an efficient and flexible broadcast authentication protocol combing μTESLA and compressed Bloom filter techniques. It also supports multiuser authentication.
Dharmapurikar, S., Krishnamurthy, P., and Taylor, D. E.	ACM Conference on Applications, Technologies, Architectures and Protocols For Computer Communications, 2003.	Longest prefix matching using bloom filters	The authors introduce the use of bloom filters for the longest prefix matching for Internet Protocol (IP) routing lookups.
Feng, Z., Matt W. M., and Lionel M. N.	IEEE International Conference on Pervasive Computing and Communications, 2006.	The Master Key: A Private Authentication Approach for Pervasive Computing Environments	The authors introduce a new entity authentication approach, combining both the advantages of the traditional master keys and multiple access tokens (using BF), for pervasive computing environments.
Artan, N. S., and Chao, H. J.	IEEE, Global Telecommunications Conference, 2005.	Multi-packet signature detection using prefix bloom filters	The authors introduce Prefix BF to detect the signatures, of the worms and malicious items, that are fragmented over the network packets and thus to prevent the epidemic.
Kang, L., and Zhenyu Z.	ACM Joint international conference on Measurement and modeling of computer systems, 2006.	Fast statistical spam filter by approximate classifications	The authors introduce use of Bloom filters to deal with the problem of memory access of Bayesian filters (spam filter), by approximating classification.

2.3.2. Using Bloom filters for Web Applications

Bloom filters can be used at the different levels of the web search/ refinement. Since this is a relatively new application area, further research is required to get the maximum benefit from the use of Bloom filters. This research area can be further divided depending on the context of the use of Bloom filters: ***membership testing*** or the less explored area ***similarity detection***. Bloom filters cannot be used for exact matching due to their false positive property. However, Bloom filters can be efficiently used for similarity detection.

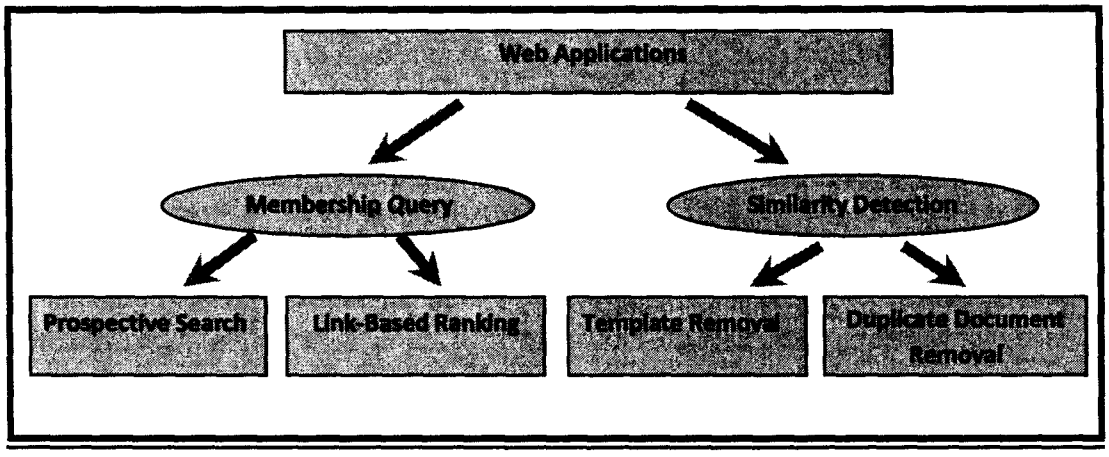

Figure 2-3: Web Applications

2.3.2.1. Membership Test Based

First is the **prospective search**. Most of the searches are retro in nature. There is another search - the prospective search by Irmak *et al* [IMSGI2006]. This can be used for news alerts, job hunting, etc. This is a keyword oriented search. The Bloom filter is used along with a hash table. During the matching phase, when the hash entries are created, the corresponding bits in the Bloom filter are also set. The overhead for this is quite low.

During the testing phase, first a lookup is done into the Bloom filter to see if there might be a hash entry for the current *QID* (integer query ID). If the answer is negative, the process is continued with the next posting otherwise a lookup is performed into the hash

table. Here, not much can be done with the Bloom filter variants. A standard Bloom filter is sufficient.

One of the basic problems in Information Retrieval on web applications is the ***ranking problem***: how to arrange the documents that satisfy a query into an order such that the documents most relevant to the query rank first. A solution is provided by **Link-based ranking** algorithms which, again, can be grouped into two classes: query-independent (in-link count or Google's famous PageRank) and query-dependent (HITS and SALSA). What can be done further is to lower the query-time cost of HITS-like ranking algorithms, i.e. algorithms that perform computations on the distance-one neighborhood graph of the results to a query [GNPmc]. The basic idea is to compute a summary of the neighborhood of each page on the web (an operation that is query-independent and thus can be done off-line, may be at index construction time), and to use these summaries at the time of the query for the approximation of the neighborhood graph of the resulting set and hence, to compute scores using this approximate graph.

Here, Bloom filters are used to store the ancestors, descendents in the neighborhood graph to facilitate the membership testing of the elements. Since, here, consistent sampling is done instead of the random one the properties relevant to the ranking algorithms are preserved in spite of using approximation of the neighborhood graph.

2.3.2.2. Similarity Detection Based

In web search, content relevance and link analysis comes next. Earlier, not much attention was paid towards this area. But then researchers got interested in web search refinement as it became the need of the time and of the industry.

Web search engines have now become a necessity as they offer a lot of convenience to the users. Therefore, their performance is always under the scrutiny and day after day better performance is expected. A necessary but also a hindrance in their efficiency is a feature called Templates. **Templates** in web sites decrease the web search engine

performance. There are other template removal methods but their main bottleneck is speed and scalability. Therefore, Chen *et al* [CYL2006] have used Bloom filters to overcome this problem. In the first phase, web segmentation is done. Then the blocks are clustered based on their layout style information. During the second phase, the word offset distribution in the block is used to measure the similarity between the contents. Here, Bloom filters are used. The evaluation is fast here as matching is only a bit wise AND operation.

Next is the improving of the search experience by **removing** or grouping **all near duplicate documents** in the results presented to the user by Jain N. *et al* [JDT2005]. There are three stages required here:

1: In the first stage content defined chunking is done to extract the document features.
2: In the second stage, these features are used as the set elements for the Bloom filter.
3: Finally, these Bloom filters are compared to detect the near-similar documents (a threshold value is already defined).

Compared to other approaches like shingling, the Bloom filter approach is faster as only bitwise ANDing is required. The second advantage is the compact size of the Bloom filter. Due to this compact size, it can be easily attached to the document tag. Although, it would have been more beneficial if a compressed Bloom filter was used; lesser data would have to be transferred.

Table 2-2: Summary of the Research on the use of Bloom filters for Web Applications

Author(s)	Conference/Journal and Year	Title of the Paper	Main Contribution
Irmak, U., Mihaylov, S., Suel, T., Ganguly, S., and Izmailov, R.	International conference on World Wide Web, ACM, 2006.	Efficient Query Subscription Processing for Prospective Search Engines	The authors introduce the Prospective search engine utilizing the bloom filters for effective membership testing.
Gollapudi, S., Najork, M., and Panigrahy, R.	research.microsoft.com, 2007.	Using Bloom Filters to Speed Up HITS-like Ranking Algorithms	The authors attempts the major problem of information retrieval and introduces a new technique, utilizing bloom filter, for reducing the query-time cost of query dependent link-based ranking algorithms
Chen, L., Ye, S., and Li, X.	ACM symposium on Applied computing, 2006.	Template detection for large scale search engines	The authors introduce a two-stage template detection method approach which combines the template detection and removal to the index building process of a search engine. The paper introduces a bloom filter based technique for finding similar sequences
Jain, N. , Dahlin, M. and Tewari, R.	International Workshop on the Web and Databases, 2005.	Using Bloom Filters to Refine Web Search Results	The authors introduce the solution to the web search efficiency problem by refining the web search results by using bloom filters for the similarity detection

2.4. Delta Bloom filter

In a scenario where an update to the Bloom filter has to be sent, these updates can either be new Bloom filters or representations of the changes between the updated filter and the original filter. Suppose there are two Bloom filters, BF1 and BF2. If an update has to be sent, another Bloom filter can be generated, which represents the update, by performing Exclusive-OR between the two Bloom filters as represented in Figure 2-4. The newly generated Bloom filter representing the update or *delta* is known as the d*elta Boom filter.* Delta Bloom filters are discussed in detail in the Section 4.1.

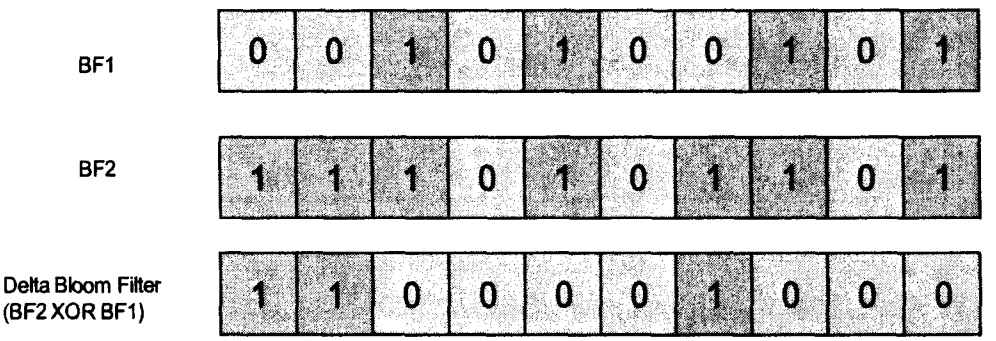

Figure 2-4: Delta Boom filter

One of the practical applications for the delta Bloom filter is sharing of caches between web proxies. Sharing of caches between web proxies is an important technique for reducing web traffic and relieving network bottlenecks. Each proxy maintains a Bloom filter representing its local cache. It also holds the Bloom filters representing caches of the other proxies. The proxies periodically broadcast updates to their cache contents [FCAB2000]. Here, d*elta Bloom filters* can be sent as the updates.

Chapter 3
DATA COMPRESSION

In the last two decades, there has been a vast improvement in the way communication takes place. Ever growing Internet, mobile and video communications have become an integral part of our lives. All these require large amounts of data to be processed, stored and transmitted. Another aspect that is common to all these, as a major enabler, is *data compression*. It would not be an exaggeration to say that if data compression was not there, then we would not be enjoying today's facilities for communication and entertainment. It would not even be practical to load images on the websites. Little do we realize that even for a long distance phone call, it is the data compression in the background present as one of the enablers.

There is an example of data compression from mid-19^{th} century in the form of Morse code introduced by Samuel Morse. Morse had observed that a telegram, represented with dots and dashes, has some letters having more frequency than others. Based on this observation he assigned shorter codes to the letters occurring more frequently (for example, '.' for e). This reduced the average time to send the message.

Data compression can range from simple removal of extra space characters to the most complex and sophisticated currently available statistical models. For example, a simple removal of all extra space characters, insertion of a single repeat character to indicate a string of repeated characters, and the substitution of the smaller bit strings for the frequently occurring characters, can reduce the original size of a text file up to 50 % [SK2000].

Data compression can be defined as the process of encoding using fewer bits than un-encoded data through the use of encoding or compression techniques. In general terms, data compression is a science of representing data in a compact form. The selection of the compression tools for a particular application depends on the characteristics of the data and application for which it is going to be used: streaming versus file; expected patterns

and regularities in the data; relative importance of CPU usage, memory usage, channel demands and storage requirements, along with other factors [MertIbm].

This chapter presents different categories of the data compression techniques as well as various encoding methods and models.

3.1. Lossless Data Compression

Data compression techniques can be classified as of two types: Lossy and Lossless.
Lossy compression involves techniques in which compressing the data and then decompressing it, retrieves data that is not exactly the same as the original, but is close enough to be useful for the application. For an example, if there is a need to compress an image representing scenery, some loss of data would not matter much. Here, limitations of human perception can be taken advantage of.

Lossless compression is a technique that reduces the size of a file without losing any data at all. That means that when a file is compressed, it will take up less space, but when decompressed, it will still have the exact same data. For instance, images representing medical data cannot afford to lose data between generation, storage, transmission and reconstruction.

For lossless data compression, X, in Figure 3-1, is exactly same as Y whereas for lossy compression schemes, X is slightly different from Y but in the allowable range. C_x represents the compressed form of X. Y is reconstructed from C_x. As a trade off to this distortion, lossy compression techniques provide higher compression rate.

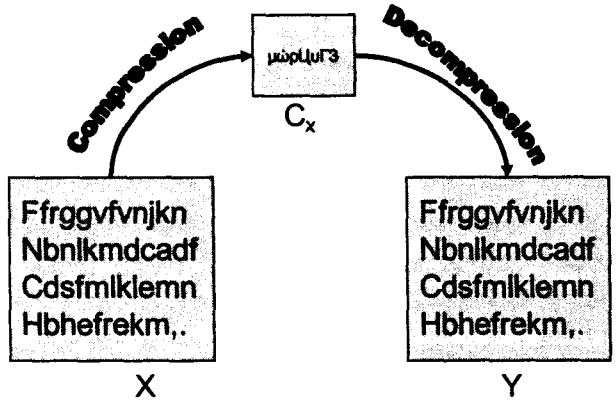

Figure 3-1: Compression and Decompression

3.2. Encoding Methods

By taking advantage of redundancy or patterns in the data, it is possible to *abbreviate* the data so as to take up less space yet maintain the ability to reconstruct a full version of the original data when required. In the encoding process, a code word, where a code alphabet $A = \{a_1, \ldots, a_n\}$, is associated to represent each source alphabet word, where a source alphabet $S=\{s_1,\ldots,s_m\}$, so as to minimize the size of the encoded data.

For lossless compression, we need *uniquely decodable* codes, that is, any given sequence code words can be decoded in one and only one way. A code is uniquely decodable if its extension is non-singular (it is a prefix code). For example, the mapping

$$C = \{a_1 \rightarrow 0, a_2 \rightarrow 10, a_3 \rightarrow 110, a_4 \rightarrow 111\}$$

is uniquely decodable. This can be demonstrated by looking at the *follow-set* after each target bit string in the map, because each bit string is terminated as soon as we see a 0 bit which cannot follow any existing code to create a longer valid code in the map, but unambiguously starts a new code.

3.2.1. Huffman Coding

In Huffman coding, the assignment of code words to source messages is based on the probabilities with which the source messages appear in the message sequence. Messages which appear more frequently are represented by short code words while messages with smaller probability map to longer code words. These probabilities are determined before the transmission begins or can be estimated while encoding in a process called *adaptive coding*. A code is said to be adaptive if the mapping from the set of messages to the set of code words changes with time. *Static Huffman coding* requires the knowledge of the probabilities of the source sequence while *Adaptive Huffman coding* requires computation of an approximation to the probabilities of occurrence at an instant, as the sequence is being transmitted. The assignment of code words to messages is based on the values of the relative frequencies of occurrence at each point in time. A message X can be represented by a short code word early in the transmission because it occurs frequently at the beginning of the sequence, even though its probability of occurrence over the total collection is low. Later, when the more probable messages begin to occur with higher frequency, the short code word will be mapped to one of the higher probability messages and X will be mapped to a longer codeword.

3.2.2. Shannon – Fano Coding

In 1960's Claude E. Shannon (MIT) and Robert M. Fano (Bell Laboratories, also taught at MIT) introduced a coding procedure to generate a binary code tree. The procedure evaluates the symbol's probability and assigns code words with a corresponding code length. Compared to other methods Shannon-Fano coding is easy to implement. In practical operation Shannon-Fano coding is not of larger importance. This is due to the lower code efficiency in comparison to Huffman coding explained above. To create a code tree according to Shannon's and Fano's algorithm, an ordered table is required providing the frequency of any symbol. Each part of this table is divided into two segments and '*0*' is added to the code words in one part while '*1*' is added to the other part. The algorithm has to ensure that the upper and the lower part of the segment have

nearly the same sum of frequencies. This procedure is repeated until only a single symbol is left.

3.2.3. Arithmetic Coding

Arithmetic coding, a special type of entropy coding, is a popular method for generating variable-length codes but follows a different approach for data compression in comparison to the static methods. It is more useful while dealing with sources with small alphabets, for example, binary sources and alphabets with highly skewed probabilities. Arithmetic coding is capable of achieving compression results which are arbitrarily close to the entropy of the source [LH1987]. Deviations which are caused by the bit-resolution of binary code trees do not occur by using this coding. In contrast to a binary Huffman code tree the arithmetic coding offers a clearly better compression rate. On the other hand, its implementation is much more complex. The most well-known paper on practical arithmetic coding algorithm is by Rissanen and Langdon [RL1979]. Arithmetic coding is also a part of the JPEG 2000 image format [JPEG2009].

The major drawback of arithmetic coding is its low speed because of the several required multiplications and divisions for each symbol. It is a general technique for coding the outcome of a stochastic process based on an adaptive model. The main idea behind arithmetic coding is to assign to each symbol an interval. Starting with the interval [0..1), each interval is divided into several subintervals, where sizes are proportional to the current probability of the corresponding symbols of the alphabet. The subinterval from the coded symbol is then taken as the interval for the next symbol. The output is the interval of the last symbol. Implementations write bits of this interval sequence as soon as they are certain.

3.3. Adaptive Methods

All of the adaptive methods are one-pass methods; only one scanning of a sequence is required. Static coding requires two passes: one pass to compute probabilities and determine the mapping, and a second pass for transmission. Thus, as long as the encoding and decoding times of an adaptive method are not substantially greater than those of a static method, the fact that an initial scan is not needed implies a speed improvement in the adaptive case. In addition, the mapping determined in the first pass of a static coding scheme must be transmitted by the encoder to the decoder. The mapping may preface each transmission (that is, each file sent), or a single mapping may be agreed upon and used for multiple transmissions. In one-pass methods, the encoder defines and redefines the mapping dynamically, during transmission. The decoder must define and redefine the mapping in sympathy, in essence, learning the mapping as code words are received.

3.4. Higher-Order Models

Data Compression techniques can also be broadly divided into two categories, the ones which use statistical techniques and the others which depend on the use of a dictionary.

Dictionary-based compression techniques generally create a dictionary (a pattern of the characters) in memory as data is scanned looking for repeated data (some implementations use a static dictionary so it does have to be built dynamically). Based on a pattern recognition process (a look-up in the dictionary), that string of data is replaced by a much shorter but uniquely identifiable string. This results in a compression of that overall data.

Archiving applications tend to use more of the dictionary-based compression techniques while statistical compression techniques are used more often for real-time applications. The reason for this selection is based on the speed of the compression and decompression. Dictionary-based compression techniques have slow compression but fast decompression speed, whereas statistical compression techniques usually have

similarly fast compression and decompression speed. Adaptive dictionary methods can be traced to the research papers by Ziv and Lempel [ZL1977] [ZL1978]. A combination of statistical and dictionary based technique is also possible.

Statistical encoding techniques determine the output on the basis of the probability of the occurrence of the input data sequence. They operate by encoding the symbols one at a time. The symbols are encoded into variable length output codes. The length of the output code varies based on the probabilities or frequencies of the symbols. Lower-probability symbols are encoded using more bits while high probability symbols are encoded using fewer bits. As such algorithms are more symmetric in nature, the compression and decompression generally requires the same amount of time. Huffman and arithmetic coding are probably the most common and widely-used statistical compression techniques [MBB2003].

3.4.1. Models

A good model for the data is useful in estimating the entropy of the source and leads to an efficient compression algorithm. A mathematical model for modeling data is usually needed to develop techniques involving manipulation of data using mathematical operations. There are many approaches for creating mathematical model for compression [Syd2000].

3.4.1.1. Probabilistic Models

The simplest statistical model for the source is with an assumption that each one of the letters generated by the source is independent of each other letter, and each of them has a particular probability of occurrence. Furthermore, the independence assumption of the letters can be used with the assignment of the probability of occurrence to each letter. The *probability model* for the source generating the letters from the alphabet $A = \{ a_1, a_2, ..., a_m\}$, is defined as $P = \{P(a_1), P(a_2), ..., P(a_m)\}$. Having a probability model, the entropy of the source can be calculated.

3.4.1.2. Markov Models

When the probability model requirement of the assumption about the independence of the letters cannot be met, there is a need to describe the dependence of the elements of the data sequence on each other. This can be achieved by a model introduced by a Russian mathematician, Andrei A. Markov and hence known as *Markov Model.*

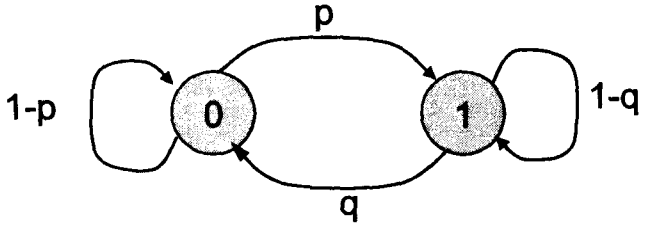

Figure 3-2: A two- state Markov model

Knowledge of the past *k* symbols is equivalent to the knowledge of the entire past history of the process. This indicates the dependence between the symbols not the form of the dependence. The use of the Markov model does not require the linearity assumption. A Markov model can be represented by a two-state diagram, for example in case of binary images, as in Figure 3-2. This Figure consists of nodes *0* and *1*, representing possible states of the system, connected by arrows, representing the rate at which the system operation transitions from one state to the other state. A state-to-state transition is characterized by a probability distribution. The probabilities of the transition arrows emanating from any state must sum to 1. Here, transition probability from state *1* to *0* is *q* and that of being in the same state *1* is 1-*q*.

3.4.2. Prediction with Partial Matching (PPM)

Some of the most effective results in data compression have been achieved by statistical source modeling in combination with arithmetic coding. Specifically, prediction with partial matching (PPM) has generated notable results. The PPM algorithm was proposed by Cleary and Witten [CW1984]. Substantial improvements and analyses have been presented since its introduction [Mitz2002]. The techniques which make use of the past history of the data being encoded to give more efficient compression are known as *predictive coding schemes*. There has been resurgence in the use of predictive coding in its many forms. The latest JPEG standard for lossless image compression uses a predictive coding algorithm [JPEG2009].

The PPM algorithm initially tries to use the largest context whose size is predetermined. In case symbol to be encoded has not been encountered in this context, an escape symbol is encoded and the algorithm tries to use the next smaller context. If, in this context the symbol is not found, then the context size is reduced further. This process continues until the context which has been previously encountered with this symbol has been found or it can be concluded that the symbol has not been encountered in any of the contexts. In this scenario, a probability $1/M$ is used to encode the symbol (M is the size of the source alphabet). For instance, while coding *h* of the string "***matching***", it is first check if the string "***match***" has been encountered before, i.e. if *h* has occurred in context "*matc*". If it has not appeared in this context, then escape is encoded and checking for the lower order context, "*atc*", takes place. If this has also not occurred then again an escape flag is transmitted and context "*tc*" is checked. In the scenario if we escape from every context then that is a special "order -1" context in which every letter has equal probability or a frequency 1.

3.4.3. Burrows – Wheeler Transformation (BWT)

The Burrows-Wheeler compression algorithm, introduced in 1994 by Michael Burrows and David Wheeler [BWT1994], is a recent development in the field of lossless data

compression. The algorithm received considerable attention because of its Lempel-Ziv like execution speed and its compression performance close to state-of-the-art PPM algorithms.

It is based on a permutation of the input sequence - the Burrows-Wheeler Transformation (BWT), also called Block Sorting, which groups symbols with a similar context close together. In the original version, this permutation was followed by a *Move to Front* (MTF) transformation and a final entropy coding (EC) stage. Later versions used different algorithms which come after the Burrows-Wheeler transform, since the stages after the Burrows-Wheeler transform have a significant influence on the compression rate too. In many approaches the MTF stage is replaced by a different stage in order to achieve a better ranking. Since the task of the MTF stage or its replacement is to transform the local structure of the BWT output into a global structure the stage is called a Global Structure Transformation (GST) as represented in Figure 3-3. Representatives of GSTs are MTF, WFC, AWAF, IF, SIF and DC. A Run Length Encoding (RLE) stage, which exists in many variations, is also common, mostly in front of the EC Stage [BWT1994].

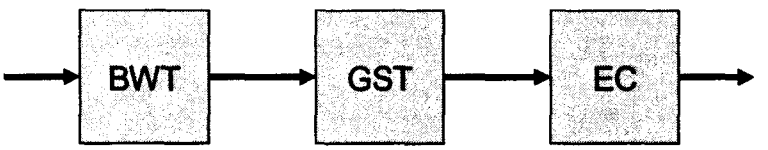

Figure 3-3: A basic Burrows - Wheeler Compression scheme

3.4.4. Stochastic Learning-based Weak Estimators (SLWE)

A new family of "weak" estimators has been introduced recently [OR2005], known as Stochastic Learning-based Weak Estimators (SLWE), and developed by using the principles of stochastic learning. The uniqueness of this new approach lies in its

capability of dealing with data coming from different and time-varying sources, making it suitable for, among other applications, a large range of files types which are transmitted or stored in computer systems. This technique is used for the proposed delta Bloom filter compression and is discussed in detail in the next chapter.

Chapter 4

PROBLEM SPECIFICATION AND SOLUTION

This chapter presents the delta Bloom filter concept in detail including the real world applications and the proposed compressed delta Bloom filter. Then the implementation of the delta Bloom filter compression by the use of a higher-order model in terms of prediction with partial matching is presented. Next, the implementation details of the delta Bloom filter compression by the use of the stochastic learning-based weak estimation method, is presented. At last the benefits of the proposed delta Bloom filter compression are presented.

4.1. Delta Bloom Filter

The Squid Web Proxy Cache is a fully featured publicly available internet caching server which is capable of handling all types of web requests on behalf of a user. A request from the user for a web resource, for instance a web page or a movie, is passed to the real web server through the caching server. When the real web server returns the requested resource to the caching server, it stores a copy of the resource in its *cache* prior to forwarding it to the user. In the future, if the user requests a copy of the *cached* resource, it is delivered from the local proxy server instead of being delivered from a real web server located far away from the proxy.

The real advantage of a proxy server lies in greatly reducing the web browsing speed as the frequently visited sites and requested resources are stored locally in the cache. The commercial profit can be gained by large organizations having a large number of internet users. On a small scale, it has the advantage for the small businesses or households having a download quota. Squid web proxy servers also have several other features like access control [SQ].

Sharing caches between web proxies is an important technique for reducing web traffic and relieving network bottlenecks. User requests to the server are handled through the proxy. Each proxy maintains a Bloom filter that represents its local cache as shown in Figure 4-1. It also holds the Bloom filters representing caches of the other proxies. These proxies periodically broadcast updates to their cache contents. These updates can either be new Bloom filters or representations of the changes between the updated filter and the original filter. These updates to the Bloom filters can be exchanged periodically, or after a certain percentage of the documents in the cache has been replaced, depending on the underlying implementation. This fairly new web cache protocol called Summary Cache was introduced by Fan *et al.* [FCAB2000]. Summary Cache is implemented in Squid v1.1.14. Also, a variation of this procedure called cache digest is implemented in Squid 1.2b20 [SQ].

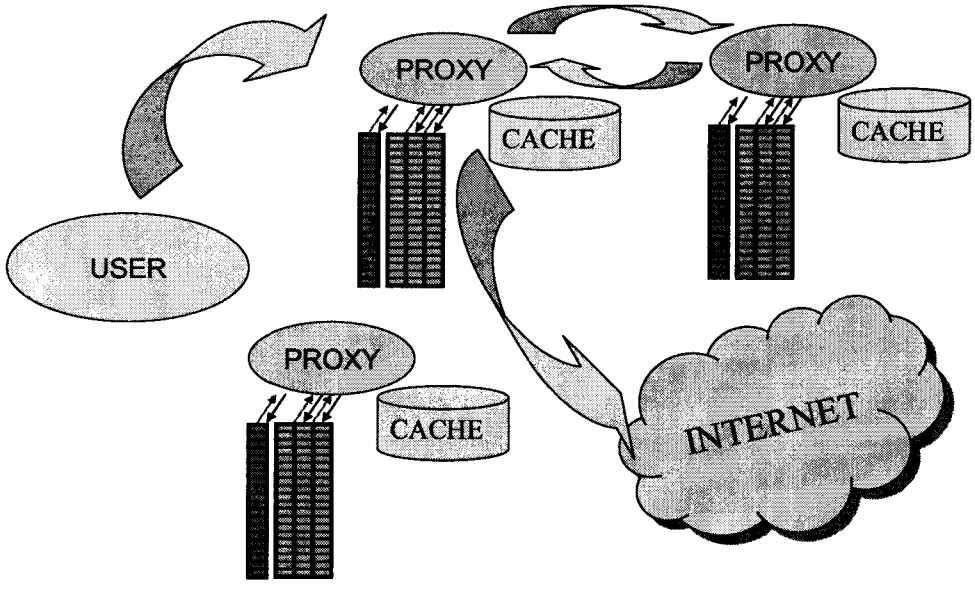

Figure 4-1: Summary Cache Representation.

The difference, or *delta*, between the updated and original filter can be represented by performing exclusive-OR of the corresponding bit arrays. As represented in the following Figure 4-2, suppose there are two Bloom filters *BF1* and *BF2* of the same size. Delta

Bloom filter is formed by performing exclusive-or operation on $BF1_i$ and $BF2_i$ and storing in dBF_i. Here, $dBF_7 = BF1_7 \oplus BF2_7$.

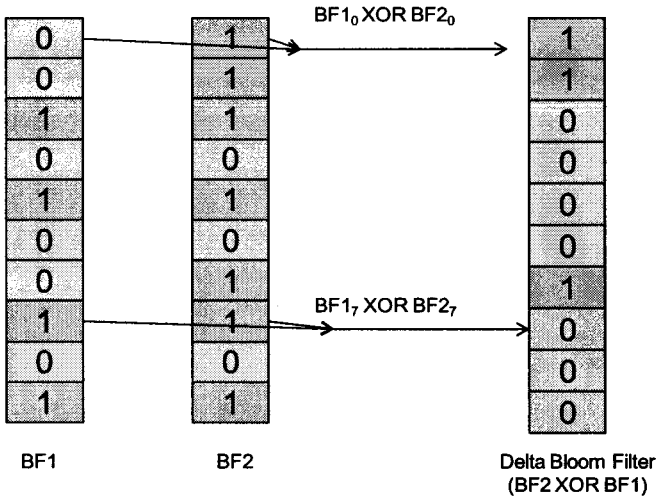

Figure 4-2: Delta Bloom Filter

The delta Bloom filter can then be compressed. Compression of the delta Bloom filter using arithmetic coding was first discussed by M. Mitzenmacher [Mitz2002].

This thesis, proposes the delta Bloom filter compression by using a fairly new technique known as stochastic learning based weak estimation (SLWE) [RO2005]. This technique is adapted for the use with the data. The results show that SLWE based compression results in much more compression gain in comparison to the benchmark arithmetic coding and also with the use of the higher order PPM based compression. The PPM algorithm has been adapted for the delta Bloom filter compression and presented in this thesis.

Due to the inexistence of the appropriate standard data benchmarks, the efficiency of the algorithm has been proved through simulation with synthetic data. This is the same approach which has been followed by M. Mitzenmacher [Mitz2002] to present their experiment results. A pseudorandom number generator is used for generating the values

for creating the Bloom filters. A seed is used with the pseudorandom generator to initialize it. This ensures that each time testing of the implemented algorithm is performed the same random number series is generated. These values are passed to k hash functions, where $2 \leq k \leq 10$. These hash functions generate a value within the range of m, the size of the Bloom filter. The bits at the corresponding index positions generated by the hash functions are set to 1. Figure 4-2 represents a Bloom filter with $m = 10$. Element a passed to four hash functions as $k = 4$. The result of the first hash function H_1 is the index 2. Therefore, the bit at position 2 is set to 1. Similarly, for element a, the bits at positions P_4, P_7 and P_{10} are set to 1.

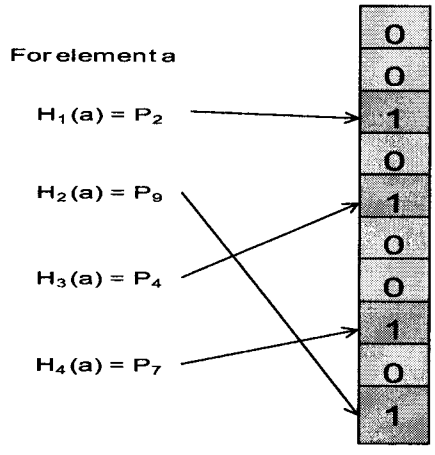

Figure 4-3: Bloom filter with 4 hash functions.

Due to the difficulty of dealing with the implementation of arithmetic coding, the maximum number of bits that would be required to be sent to the decoder for decompression are calculated.

4.2. Adaptive Unigram Model

The adaptive unigram model tracks the count of the bytes encountered before and based on the frequency count estimates the probability for the byte *b*. The model updates counters after each byte, and hence it is *adaptive*. It makes use of strings of length 1, that is why, it is known as *unigram*. Each of the byte's counters has an initial value of 1 based on the principle of Laplace smoothing. If the sum of the counts exceeds the maximum, the counts are rescaled by dividing them by 2 and rounding up to at least 1. The adaptive unigram model with the arithmetic coding is the benchmark compression method for the comparisons in the thesis. The results of the proposed delta Bloom filter compression using PPM and SLWE are compared with adaptive unigram based compression.

4.3. Prediction with Partial Matching (PPM)

As discussed in the third chapter, prediction with partial matching is an adaptive model that can be used with the entropy-based encoding technique. Here, PPM is used with arithmetic coding. This implementation can be found at http://www.data-compression.info/Algorithms/AC/. For this thesis, it has been adapted for the use depending on the data being compressed. The thesis presents a new method that involves delta Bloom filter compression using a PPM model.

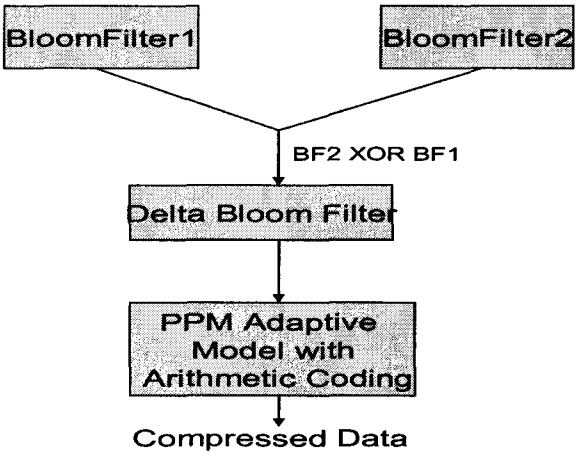

Figure 4-4: Delta Bloom filter compression using PPM.

Once the two Bloom filters are created, the exclusive-OR operation is performed on them. Exclusive-OR is a logical operator which results in a value *1* or *true* when exactly one of the operands is true. Here, exclusive-OR results in a value 1 to denote an update or delta to the Bloom filter. This operation results in the generation of the delta Bloom filter. Then, the data is converted to the byte format using a shift operator as the PPM model deals with the bytes. This data is passed to the PPM model and then encoded using arithmetic coding for the compression. For encoding, the data is accessed in the bit format.

PPM Algorithm:

Repeat

Read next symbol, s.

Let $d_K, d_{(K-1)}, \ldots, d_1$ be the preceding K characters.

Set the context size, k, to the maximum, K.

While $(d_k, \ldots, d_1)$ has not been seen previously:

    Set $k \leftarrow k - 1$.

While $k \geq 0$ and c is not in context $(d_k, \ldots, d_1)$:

    Transmit an escape flag using context $(d_k, \ldots, d_1)$.

    Set k to $k - 1$.

If $k \leftarrow -1$: Transmit symbol s using the special "order -1" context. Set $k = 0$.

Else Transmit s using context $(d_k, \ldots, d_1)$.

While $k \leq K$:

    If context $(d_k, \ldots, d_1)$ does not exist, create it.

    Increment the count for s in context $(d_k, \ldots, d_1)$.

    Set $k \leftarrow k + 1$.

Until end of file.

For decompression, the compressed data is passed to the arithmetic decoder which using the PPM model reconstructs the original data.

The algorithm for delta Bloom filter compression using PPM is as follows:

Start
Get Bloom filters BF1 and BF2
if BF1 is not equal to BF2
    calculate BFX ←BF2 XOR BF1
end if
convert BFX to byte array
use PPM model with the given context size, *K,* to provide an estimate for the probability for the n^{th} byte
encode data with cumulative probabilities
End

4.4. Stochastic Learning-based Weak Estimation (SLWE)

As discussed earlier, two Bloom filters are generated through simulation. Pseudorandom number generator is used for generating the values for creating the Bloom filters. These values are passed to k hash functions, where $2 \leq k \leq 10$. These hash functions generate a value within the range of m, the size of the Bloom filter.

Once the two Bloom filters are created, the exclusive-OR operation is performed between them. This operation results in the generation of the delta Bloom filter representing the updates to the original Bloom filter. Then Stochastic Learning-based Weak Estimation is used for updating the probabilities as represented in the following Figure 4-5.

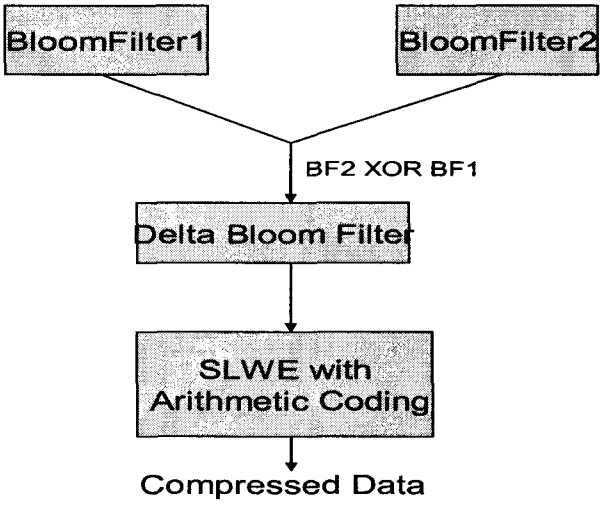

Figure 4-5: Delta Bloom filter compression using SLWE.

Binomial distribution is a discrete probability distribution of the number of successes in a sequence of n independent success/failure observations, each of which yields success/failure with probability p. Such a success/failure experiment is also called a Bernoulli trial. Therefore, it can be said that a binomial random variable is characterized by the number of the Bernoulli trials and the parameter characterizing *each* Bernoulli trial. It is assumed that the number of observations is the number of trials, and the stochastic learning methods are used to estimate the *Bernoulli* parameter for each trial. Suppose that X is a binomially distributed random variable, having the value of "0" or "1". It is an assumption that X obeys the distribution S, where $S = [s_1, s_2]^T$ where T denotes the transpose of a vector.

$$X = \text{"0" with probability } s_0$$
$$= \text{"1" with probability } s_1, \text{ where, } s_0 + s_1 = 1.$$

Let $x(n)$ be a concrete realization of X at time "n". The goal is to estimate S, i.e., s_i for $i = 0, 1$. This is achieved by maintaining a running estimate

$P(n) = [p_0(n), p_1(n)]^T$ of S, where $p_i(n)$ is the estimate of s_i at time "n", for $i = 0, 1$.

Then, the value of $p_0(n)$ is updated as per the following rule:

$$P_0(n+1) \leftarrow \lambda p_0(n) \text{ if } x(n) = 1 \quad (1)$$

$$\leftarrow 1 - \lambda p_1(n) \text{ if } x(n) = 0 \quad (2)$$

Where λ is a user defined parameter and $p_1(n+1) \leftarrow 1 - p_0(n+1)$; λ is used to learn the probability of the next symbol based on the probability of the previous symbol.

The above explained probability updating method is utilized by an entropy based encoding technique called arithmetic coding. This method for probability updating is invoked by the encoding and decoding algorithm.

The algorithm followed for the delta Bloom filter compression using SLWE is as follows:

Start
Get Bloom filters BF1 and BF2
if BF1 is not equal to BF2
    BFX ←BF2 XOR BF1
end if
set count to 0
for each chosen value of λ
    update the probabilities for each symbol in BFX
    calculate number of bits used to encode each symbol in BFX by its information amount, $I_n \leftarrow \lceil -\log_2 p_i\ (n) \rceil$
    Sum the number of bits used to encode all the symbols
    $I_{tot} \leftarrow I_{tot} + I_n$
    count ←count + 1
    end for loop
calculate the number of bits required to send λ, $bl \leftarrow \lceil \log_2 counter \rceil$
calculate total number of bits required to represent compressed data, $bitsn \leftarrow I_{tot} + bl + 2$, and round off *bitsn*.

calculate compression gain s ←(1 - bitsn) / 100
End

4.4.1. Notes on the Implementation

Due to the difficulty of dealing with the implementation of the arithmetic coding, the maximum number of bits that would be required to be sent to the decoder for decompression are calculated.

It is assumed that the symbol at time n, s_i, is encoded using the number of bits determined by its information amount that is $-\log_2 p_i\ (n)$. This also assumes that all the symbols occur independently of each other. Hence, the whole sequence, X, can be encoded by using the following number of bits:

$$\lceil -\log_2 \prod_{n=1}^{M} p(x(n)) \rceil$$

$$= \lceil -\sum_{n-1}^{M} \log_2 p(x(n)) \rceil$$

As the bits to be transferred cannot be represented as a fraction, the total information amount is rounded off.

On the other hand, the SLWE needs the parameter λ to learn the probability of the next symbol based on the probability of the previous symbol. This parameter λ is optimized based on the knowledge of the previous experiments [RO2005]. The range that is used for the proposed method is $0.9900 \leq \lambda \leq 0.9999$. This gives a total 100 different values for λ. For decompression, the decoder requires to know the value of λ in order to fully recover the original data. The number of bits required to send the values of λ can be calculated as $\lceil \log_2 100 \rceil$, that is 7 bits. Again, this value needs to be rounded off in order to avoid getting a fraction as the number of bits.

Arithmetic coding has lower and upper bounds for a number of bits sent to the output based on the information amount, entropy and the statistical data of the symbols predicted by the SLWE. These bounds on the information amount can be derived as follows.

Upper and lower bounds on the number of bits required to be sent [Syd2000]:

The number of bits $l(x)$ required to encode the entire sequence x, with enough accuracy such that the codes for different values is unique is

$$l(x) = \left\lceil \log \frac{1}{p(x)} \right\rceil + 1 \qquad (1)$$

where $P(x)$ is the probability of x.

Since, $l(x)$ is the number of bits required to encode *entire* sequence x, the average length of an arithmetic code for a sequence of the length m, I_{A^m} , is given by

$$I_{A^m} = \sum P(x) l(x) \qquad (2)$$

$$= \sum P(x) \left[\left\lceil \log \frac{1}{P(x)} \right\rceil + 1 \right] \qquad (3)$$

$$\leq \sum P(x) \left[\left\lceil \log \frac{1}{P(x)} \right\rceil + 1 + 1 \right] \qquad (4)$$

$$= -\sum P(x) \log P(x) + 2 \sum P(x) \qquad (5)$$

$$= H(X^m) + 2 \qquad (6)$$

Given that the average length is always greater than the entropy, the lower and upper bounds on I_{A^m} are

$$H(X^m) \leq I_{A^m} \leq H(X^m) + 2 \qquad (7)$$

Since the equation (7) gives an upper bound, the number of bits needed to encode a sequence of the length m will not exceed 2 bits from the number of bits determined by its information amount.

Therefore, the number of bits required to send the compressed data, two bits to cover the upper limit for arithmetic coding and the number of bits required to send the value of λ, which is 7, are added together to estimate the size of the output. All of these will make sure that the original input will be completely recovered from the compressed data and, hence, achieving lossless compression.

4.5. Benefits of Delta Bloom Filter Compression

The proposed methods will provide substantial compression gain to applications relying on heavy use of the bandwidth and distributed computing environment relying on frequent updates of the underlying data. In a scenario where Bloom filter is not just a data structure but also a message which needs to be transmitted from one location to another, it is beneficial to compress it especially where frequent updates of this Bloom filter are required to be transmitted. It will reduce the amount of data transmitted. If any of the bits of the Bloom filter are incorrectly represented due to the loss of data during compression, the element which is a member of the set might be incorrectly represented as not a member. This will defy the basic property of the Bloom filter which states that the false negative rate is not a characteristic of a Bloom filter. To ensure this does not happen, for the proposed methods, lossless compression technique is used to reduce the amount of data and there would be no loss of information due to compression. Further, using SLWE as proposed in our method, probabilities of the source symbols can be adaptively updated while being encoded requiring only one pass as opposed to the static coding method that would have required two passes.

The proposed methods are tested for the efficiency in terms of the compression gain, scalability in terms of the dependence on the size of Bloom filter and adaptability in terms of the number of hash functions used, delta change in the Bloom filter content, and

its dependence on the parameter λ. This ensures that the proposed methods perform better than the benchmark zero-order arithmetic coding for different requirements of the applications.

Chapter 5

IMPLEMENTATION AND EXPERIMENTS

This chapter presents the details of the experiments. The details of the case of delta Bloom filter compression using SLWE with arithmetic coding method are presented here. Adaptive k^{th} order prediction with partial matching model with arithmetic coding for delta Bloom filter has also been implemented. Delta Bloom filter compression using adaptive unigram with arithmetic coding acts as the benchmark to compare the performance of the proposed delta Bloom filter compression using the higher-order models with arithmetic coding, including PPM and the Stochastic Learning-based Weak Estimation (SLWE) method. The results show that further improvement in the compression gain is possible by applying the proposed compression methods.

In this chapter, first, the implementation issues and the experiment set-up are discussed. Then the results of the experiments are discussed. The proposed delta Bloom filter compression methods are tested for scalability by varying the size of the Bloom filter. The results of the experiments are compared with the varying number of the hash functions and the percentage of change between the Bloom filters to test for the adaptability. This chapter also presents the impact of varying λ, a parameter required for SLWE, on the compression gain. At the end of the chapter, the findings are compared and concluded with positive results.

5.1. Implementation Considerations

We have to deal with the different formats of the data for the two implementations. Prediction with partial matching uses data in the byte format whereas Stochastic learning-based weak estimation requires data to be in bit format. The implementation needs to be carried out so as to process the data in the required format specific to each model. Then, for the comparison between the efficiency of the two methods, the compression gain is

used. The model which has a higher compression gain or the percentage of the space savings, represented as S, is better.

Secondly, the values of λ need to be passed to the decoder as this value will be required for the reconstruction of the data from the compressed data. As explained in Chapter 4, the number of bits required to be sent to the decoder are calculated. There will be a small overhead for sending these data.

5.2. Experimental Setup

All the implementations are done in Java on Eclipse IDE on a Windows XP operating system with 760 MB of RAM, on Intel® Celeron M® 1.6 GHz processor.

For the experiments and comparisons, the implementation includes the following three schemes:

- Benchmark: the adaptive unigram model with arithmetic coding for delta Bloom filter (base for comparison).
- Adaptive k^{th} order PPM model with arithmetic coding for the delta Bloom filter.
- SLWE with arithmetic coding for the delta Bloom filter.

The basic code for the adaptive unigram and PPM models were obtained from the compression data repository [ACcomp]. In this thesis, a new method for delta Bloom filter compression is proposed using PPM. After the implementation, the code is tested with Bloom filters of various sizes, m, with different number of hash functions used to create a Bloom filter, k, with different percentage of changes for the delta Bloom filter, c, and with λ, the parameter used in SLWE values.

First, two Bloom filters of the same size using random numbers are generated. For this a pseudorandom number generator in java is used. A specific seed is used to initialize the

pseudorandom number generator. This ensures that each time the code is tested the same random number series are generated. For creating the Bloom filter, in the initial experiments, two hash functions are used. The general hash function library is used from http://www.partow.net and modified to suit the data requirements. The hashing algorithm md5, which is available in the java library, can also be used. However, this could be only one of the k hash functions.

The size of the Bloom filter, m, is set to 140,000 bits and two hash functions are used, k=2. The percentage difference between the two Bloom filters, c, is set to 5%. This setup is followed for all initial tests of the implemented methods. This is the setup discussed by Mitzenmacher for compressed Bloom filters [Mitz2002]. Therefore, to ensure that the proposed method performs with the same settings, initially, we tested our code with them.

At first, two Bloom filters are generated as discussed earlier for the initial testing and the delta Bloom filter is created by Exclusive ORing the two Bloom filters. As discussed earlier, this Delta Bloom filter represents the update in the Bloom filter. Then the delta Bloom filter is passed to the adaptive unigram model, PPM and SLWE, based on which the arithmetic coding is carried out.

5.3. Experimental Results and Observations

In this section, firstly, the results of the proposed delta Bloom filter compression using SLWE are presented. Secondly, the results of the delta Bloom filter compression using PPM are presented. The proposed methods are tested with the Bloom filters of various sizes, m, with different numbers of hash functions used to create a Bloom filter, k, with different percentages of changes for the delta Bloom filter, c and with λ, the parameter used in SLWE and tested only for SLWE values. The results have been organized in a tabular format, and then to show the trend of the results by varying the factor values is represented through the graphs

5.3.1. Results for the SLWE based Delta Bloom filter compression

In this section, we discuss the results of the experiments performed for testing the proposed delta Bloom filter compression using the SLWE technique. The pattern followed by varying the size of the Bloom filter, number of hash functions used, percentage of change between the Bloom filters and the effect of the parameter λ on our proposed method are studied and presented. It is found that the proposed methods follow a consistent pattern based on the above mentioned factors.

5.3.1.1. Size of the Bloom filter

i) The delta Bloom filter compression is tested for different sizes of the Bloom filter. The size of the Bloom filter is in the range from 4 to 10Mb. Four hash functions are used to create the Bloom filters and the percentage of change between the Bloom filters is 5%.

In Table 5-1 the results of the experiments are summarized. Table 5-1 shows the effect of the size of the Bloom filter and λ on the compression gain. For example, if a 5 Mb Bloom filter is compressed by using $\lambda = 0.9994$, a maximum compression gain of 86.98238% is achieved. It also shows that if a larger size Bloom Filter is used then the compression gain is higher. The compression gain increases with the size of the Bloom filter. Compression gain also depends on the values of λ but one particular value for λ does not result in the best result for all sizes of the Bloom filters.

Lambda Values	Bloom Filter Size						
	4Mb	5Mb	6Mb	7Mb	8Mb	9Mb	10Mb
0.999	86.716	86.976	87.487	88.121	88.711	89.222	89.732
0.9991	86.718	86.978	87.490	88.124	88.713	89.225	89.735
0.9992	86.720	86.980	87.492	88.127	88.716	89.227	89.738
0.9993	**86.721**	86.982	87.494	88.129	88.718	89.230	89.740
0.9994	86.721	**86.982**	87.495	88.130	88.720	89.232	89.743
0.9995	86.719	86.982	**87.495**	**88.131**	**88.721**	89.233	89.744
0.9996	86.715	86.979	87.493	88.130	88.720	**89.233**	**89.744**
0.9997	86.705	86.973	87.488	88.126	88.717	89.231	89.743
0.9998	86.683	86.956	87.474	88.114	88.707	89.222	89.735
0.9999	86.609	86.897	87.426	88.072	88.671	89.190	89.706

Table 5-1: Effect of the size of the Bloom filter (4Mb $\leq m \leq$ 10Mb) on the SLWE based compression

The graph in Figure 5-1 has two variables, size of the Bloom filter and the parameter λ. It shows the effect of the size of Bloom filter on the compression gain achieved by SLWE-based delta Bloom filter compression for a specific value of λ. It can be noticed that the graph follows a particular trend of consistent increase in compression gain with the increase in the size of the Bloom filter. It also shows that the difference between compression gain achieved by varying values of λ for a particular size of the Bloom filter is very close.

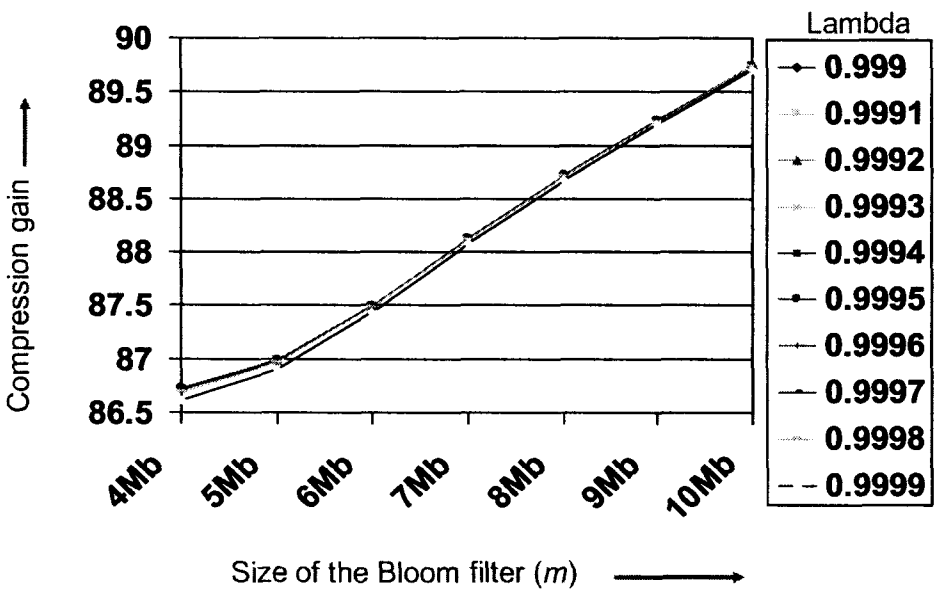

Figure 5-1: Effect of the size of Bloom filter on the SLWE-based compression

ii) Delta Bloom filter compression has also been tested on large size Bloom filters. During these experiments, the size of the Bloom filter is set in the range 5 to 40Mb. Four hash functions are used to create the Bloom filters and the percentage of change between the Bloom filters is 5%. Table 5-2 shows the effect of the size of the Bloom filter and λ, a parameter used for SLWE technique, on the compression gain. For instance, if a 30 Mb Bloom filter is compressed by using λ = 0.9997, a maximum compression gain of 94.82777 % is achieved. It also shows that if a larger size Bloom Filter is used, then the compression gain is more. The compression gain increases with the size of the Bloom filter. The compression gain depends on the values of λ and one particular value of λ does not result in the best result for all the sizes of the Bloom filters.

Lambda values	Bloom filter size							
	5Mb	10Mb	15Mb	20Mb	25Mb	30Mb	35Mb	40Mb
0.9990	86.976	89.732	91.769	93.140	94.108	94.806	95.374	95.804
0.9991	86.978	89.735	91.773	93.144	94.112	94.810	95.379	95.808
0.9992	86.980	89.738	91.776	93.147	94.115	94.814	95.383	95.812
0.9993	86.982	89.740	91.779	93.150	94.118	94.817	95.387	95.816
0.9994	**86.982**	89.743	91.781	93.153	94.122	94.821	95.390	95.819
0.9995	86.982	89.744	91.784	93.156	94.124	94.824	95.393	95.822
0.9996	86.979	**89.744**	**91.785**	93.158	94.127	94.826	95.396	95.825
0.9997	86.973	89.743	91.785	**93.158**	**94.128**	**94.828**	95.398	95.827
0.9998	86.956	89.735	91.781	93.156	94.127	94.827	**95.398**	**95.828**
0.9999	86.897	89.706	91.762	93.142	94.116	94.819	95.391	95.822

Table 5-2: Effect of the size of the Bloom filter (5Mb $\leq m \leq$ 40Mb) on the SLWE-based compression

As noticed from Table 5-2, there is a consistent trend of the increase in the compression gain with the increase in the size of the Bloom filter. However, we cannot choose one specific value for λ. The graph in Figure 5-2 makes the result trend visually comprehensible. The two parameters here are the size of the Bloom filter and λ, the parameter used for SLWE-based compression. The graph represents the effect of these two variables on the compression gain achieved by the proposed delta Bloom filter

compression method. It shows that if a larger size Bloom Filter is used, then the compression gain is increased.

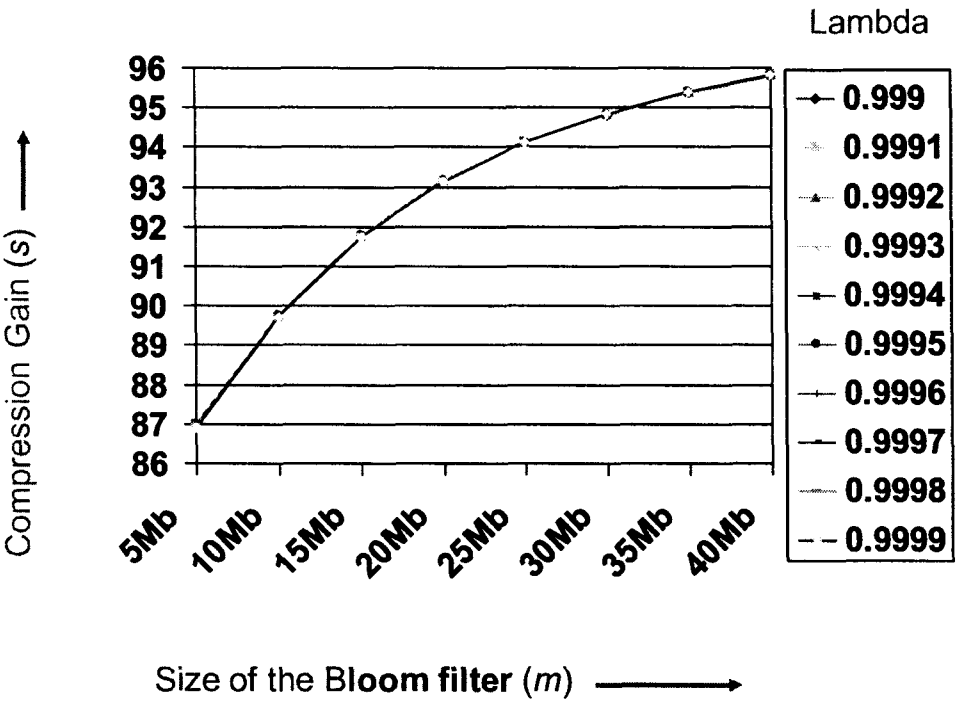

Figure 5-2: Effect of the size of the Bloom filter on SLWE-based compression.

5.3.1.2. Number of Hash Functions Used

The experiments were performed for the different number of hash functions used to create the Bloom filter. By increasing the number of hash functions, we can test the adaptability of the proposed delta Bloom filter compression. Initially, Bloom filters of size 140Kb were used. Table 5-3 summarizes the findings. The number of hash functions used, k, is in the range 2 to 10 and λ is in the range 0.99 to 0.999. It can be noticed that as the number of the hash functions used increases, the compression gain decreases. By using two hash functions for generating a Bloom filter of size 140Kb, a compression gain of 94.19 % can be achieved.

Lambda values	Number of Hash functions, k								
	2	3	4	5	6	7	8	9	10
0.990	93.821	92.117	90.758	89.591	88.566	87.890	87.237	87.042	86.716
0.991	93.898	92.183	90.816	89.646	88.618	87.940	87.285	87.090	86.762
0.992	93.972	92.246	90.874	89.699	88.669	87.989	87.332	87.137	86.807
0.993	94.041	92.305	90.925	89.747	88.715	88.034	87.375	87.179	86.849
0.994	94.101	92.355	90.970	89.790	88.756	88.073	87.412	87.217	86.886
0.995	94.151	92.398	91.007	89.825	88.791	88.106	87.444	87.249	86.917
0.996	94.187	92.428	91.033	89.850	88.815	88.128	87.466	87.271	86.938
0.997	**94.199**	**92.437**	**91.039**	**89.855**	**88.822**	**88.133**	**87.470**	**87.276**	**86.942**
0.998	94.163	92.401	91.004	89.820	88.788	88.098	87.434	87.242	86.908
0.999	93.954	92.199	90.807	89.627	88.600	87.911	87.248	87.057	86.727

Table 5-3: Effect of the Number of the Hash Functions on the SLWE based compression

The graph in the Figure 5-3 represents the data presented in Table 5-3.The two variables here are the number of hash functions used to create a Bloom filter and λ. It shows the trend of the decrease in the compression gain with increase in the number of hash functions. As it can be noticed from the graph, the compression gain with respect to λ

values is very close but consistently decreases with the number of hash functions used to create Bloom filters.

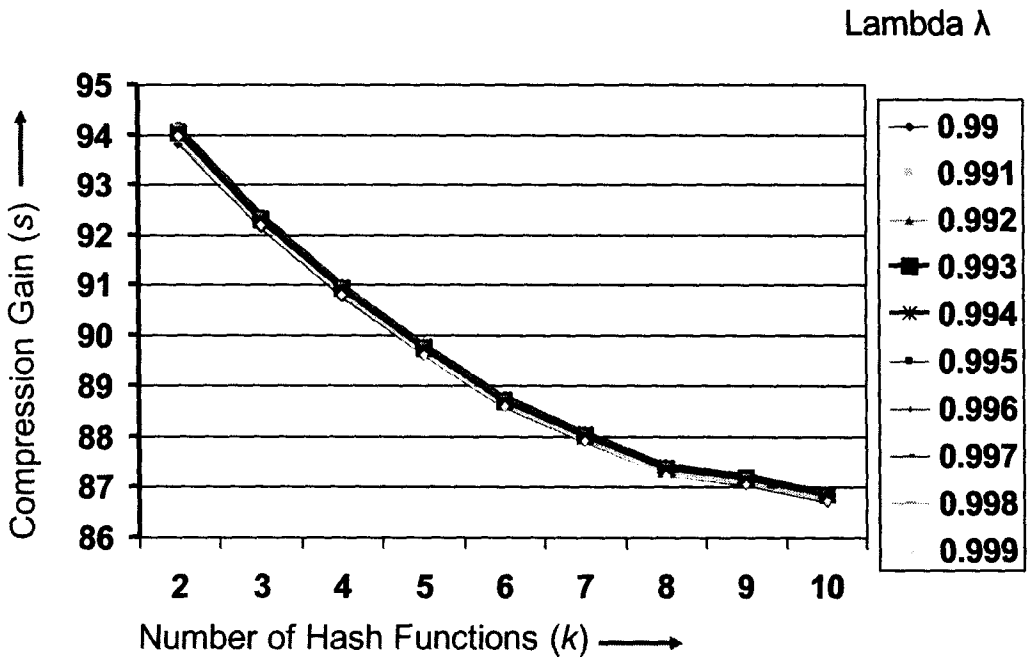

Figure 5-3: Effect of the Number of the Hash Functions on SLWE-based compression.

5.3.1.3. Percentage of Change (c) Between the Two Bloom Filters

The experiments were performed to test the effect of the change between two Bloom filters on the compression models. For this comparison, Bloom filters of size 140Kb are created using two hash functions. Table 5-4 summarizes the results. The experiments were performed for the percentage of change between the two Bloom filters ranging from 1% to 10 %. It can be noticed that the compression gain decreases with the increase in c.

Lambda Values	% of change , c, between the Bloom filters									
	1	2	3	4	5	6	7	8	9	10
0.990	97.456	96.503	95.595	94.691	93.821	92.947	92.106	91.312	90.567	89.783
0.991	97.574	96.608	95.690	94.777	93.898	93.017	92.171	91.373	90.626	89.838
0.992	97.691	96.710	95.781	94.859	93.972	93.085	92.233	91.432	90.681	89.890
0.993	97.805	96.806	95.867	94.935	94.041	93.147	92.290	91.485	90.732	89.938
0.994	97.911	96.895	95.944	95.004	94.101	93.201	92.340	91.531	90.777	89.979
0.995	98.008	96.974	96.011	95.063	94.151	93.245	92.380	91.570	90.814	90.013
0.996	98.090	97.036	96.062	95.106	94.187	93.277	92.408	91.596	90.839	90.037
0.997	**98.144**	**97.070**	**96.086**	**95.125**	**94.199**	**93.285**	**92.415**	**91.603**	**90.844**	**90.040**
0.998	98.141	97.046	96.055	95.092	94.163	93.248	92.378	91.566	90.809	90.003
0.999	97.930	96.823	95.836	94.879	93.954	93.043	92.176	91.368	90.612	89.808

Table 5-4: Effect of % of change between two Bloom filters, c, and λ on SLWE-based delta Bloom filter compression for m = 140Kb.

The graph in the Figure 5-4 represents the experiment results summarized in Table 5-4. The two variables here are percentage change between two Bloom filters and the parameter λ. There is a consistent decrease in the compression gain with the increase in the percentage change between the two Bloom filters for a specific value of λ. The percentage of change between two Bloom filters denotes the update to the original Bloom filter. Therefore, if there is a need to send the updates very frequently, then c can be chosen as 1%. If a very frequent exchange of updated data is not required, then c = 5 can be chosen. This will cost a slight decrease in the compression gain but frequent transmission cost can be reduced.

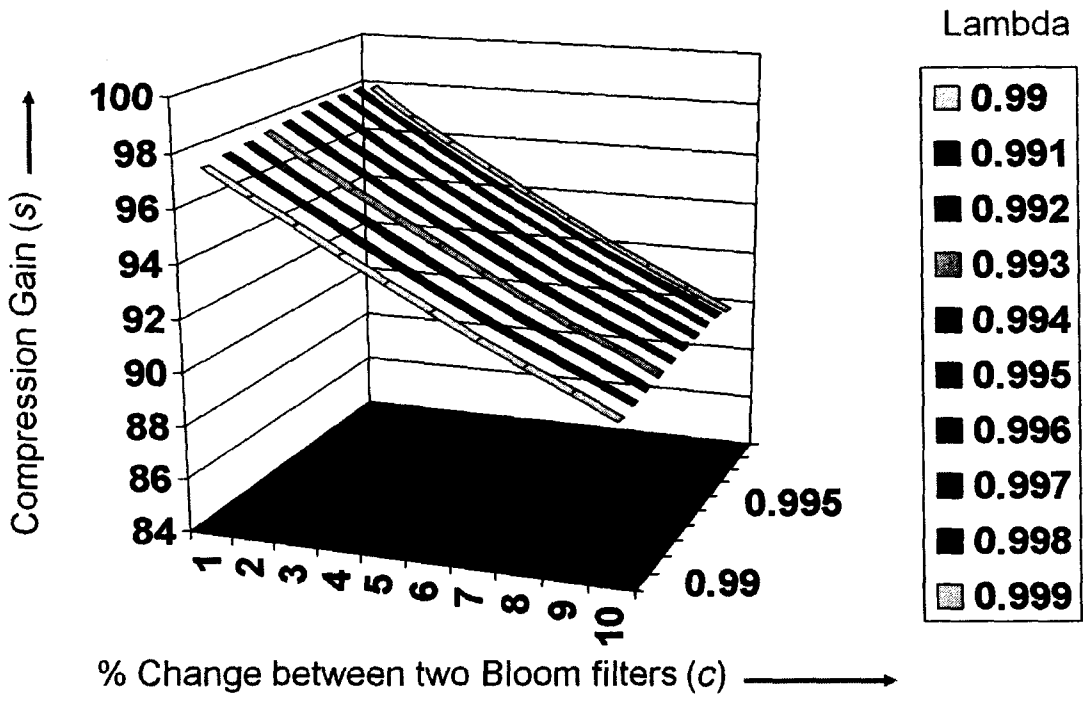

Figure 5-4: Effect of percentage of change between two Bloom filters, c, and λ on the SLWE-based delta Bloom filter compression for m = 140Kb.

5.3.1.4. User Defined Parameter λ for SLWE based Delta Bloom Filter Compression

In this case, the experiments were performed for the compression methods with a 140Kb Bloom filter generated using two hash functions and with 5% change between the Bloom filters. Table 5-5 summarizes the results. It can be noticed that the value of the parameter λ affects the compression gain achieved using the SLWE method.

Comp. Model	Lambda Value									
	0.990	0.991	0.992	0.993	0.994	0.995	0.996	0.997	0.998	0.999
SLWE	93.821	93.898	93.972	94.041	94.101	94.151	94.187	**94.199**	94.163	93.95

Table 5-5: Effect of λ on SLWE-based Delta Bloom filter compression.

The graph in the Figure 5-5 presents the effect of varying λ on the compression gain achieved by SLWE-based delta Bloom filter compression. It can be noticed that it does not follow a specific trend. The compression gain achieved at first consistently increases for $0.990 \leq \lambda \leq 0.997$. After this it starts to decrease till the maximum value set for the experiments.

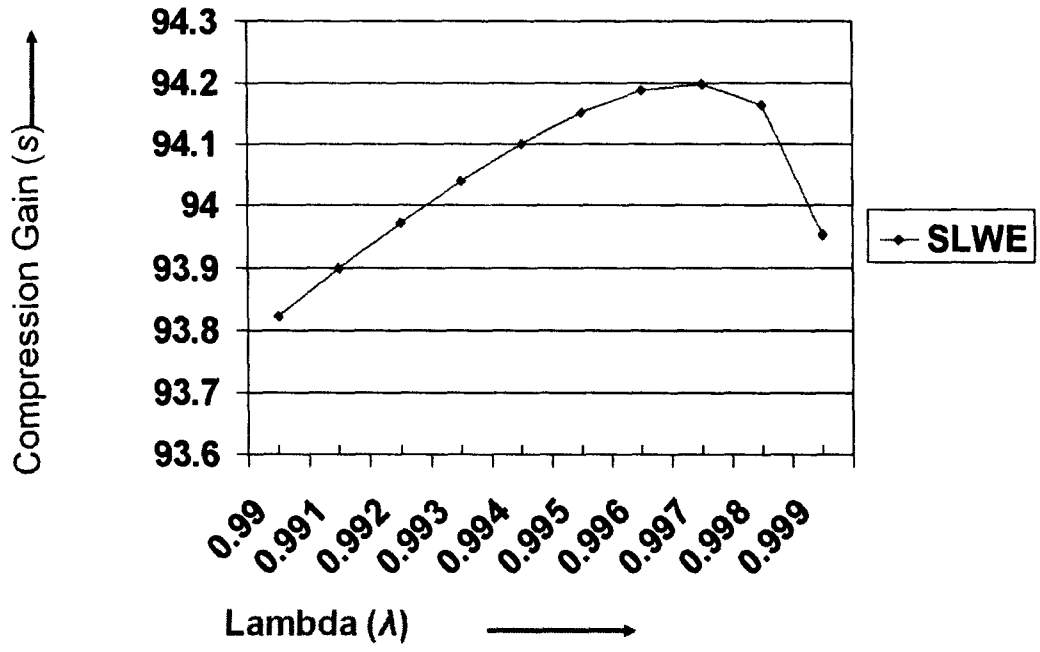

Figure 5-5: Effect of varying λ on SLWE-based delta Bloom filter compression for m=140Kb.

5.3.2. Comparison Between the SLWE-based and PPM models for Delta Bloom Filter Compression

For the comparison between the results of delta Bloom filter compression using different compression methods, the same setup is used as described in earlier sections. The tests on all the methods are performed on the same data. The cross validation of the process is done in the case of the method using PPM model by testing whether the Bloom filter reconstructed after compression is the same as the original Bloom filter. For the delta Bloom filter compression using SLWE, cross validation is not required, as we are calculating the bits required to send the compressed data and then the compression gain based on this data. In the previous section, the effect of varying the size of the Bloom

filter, number of the hash functions, percentage of change between the Bloom filters and that of the parameter λ, on our proposed delta Bloom filter compression using SLWE were seen. In this section, the comparison is performed between the benchmark adaptive unigram method, SLWE-based delta Bloom filter compression, and the one based on the higher-order model PPM.

5.3.2.1. Size of the Bloom Filter (m)

The size of the Bloom Filer plays a major role in compression. The larger the data for training the model the better the compression achieved is. The proposed method of the delta Bloom filter compression using SLWE consistently works better than the rest of the compression methods in terms of compression gain as shown in Table 5-6. For this comparison larger sizes of the Bloom filters are considered, in the range from 10 to 40 Mb, with four hash functions and percentage of change of 5%. In the table, the uniform model represents no compression of the data. It can be noticed that the SLWE-based delta Bloom filter compression provides consistent better compression gain than the unigram and the PPM models. The difference between the compression gains achieved by SLWE-based compression for a specific size of Bloom filter is not large but consistently better than the other models for all the sizes considered during the experiments.

Compression Models	Bloom filter size (Mb)						
	10	15	20	25	30	35	40
Uniform	-0.070	-0.070	-0.070	-0.070	-0.070	-0.070	-0.070
Unigram	89.680	91.719	93.088	94.057	94.756	95.325	95.753
PPM(0)	89.166	91.182	92.532	93.484	94.172	94.730	95.151
PPM(1)	88.590	90.866	92.396	93.484	94.271	94.909	95.390
PPM(2)	88.559	90.810	92.336	93.419	94.199	94.836	95.317
PPM(3)	88.498	90.743	92.269	93.352	94.133	94.769	95.252
PPM(4)	88.416	90.662	92.187	93.270	94.055	94.693	95.177
PPM(5)	88.325	90.579	92.106	93.194	93.981	94.621	95.104
PPM(6)	88.243	90.496	92.030	93.119	93.903	94.546	95.031
PPM(7)	88.157	90.417	91.955	93.043	93.829	94.475	94.962
PPM(8)	88.070	90.345	91.882	92.973	93.761	94.405	94.894
PPM(10)	87.900	90.200	91.740	92.833	93.619	94.266	94.758
SLWE	**89.744**	**91.785**	**93.158**	**94.128**	**94.828**	**95.398**	**95.828**

Table 5-6: Effect of the size of the Bloom filter (m) on the compression gain with respect to different compression models.

The graph in the Figure 5-6 represents the data from Table 5-6 for the effect of the size of the Bloom filter on the compression gain achieved by different compression models. It shows that the size of the bloom filter is one of the factors on which delta Bloom filter compression is dependent and considering this factor, SLWE-based compression consistently performs better than the unigram and the proposed PPM-based delta Bloom filter compression methods. This also proves that the proposed SLWE-based method is scalable while consistently achieving better results than the other methods.

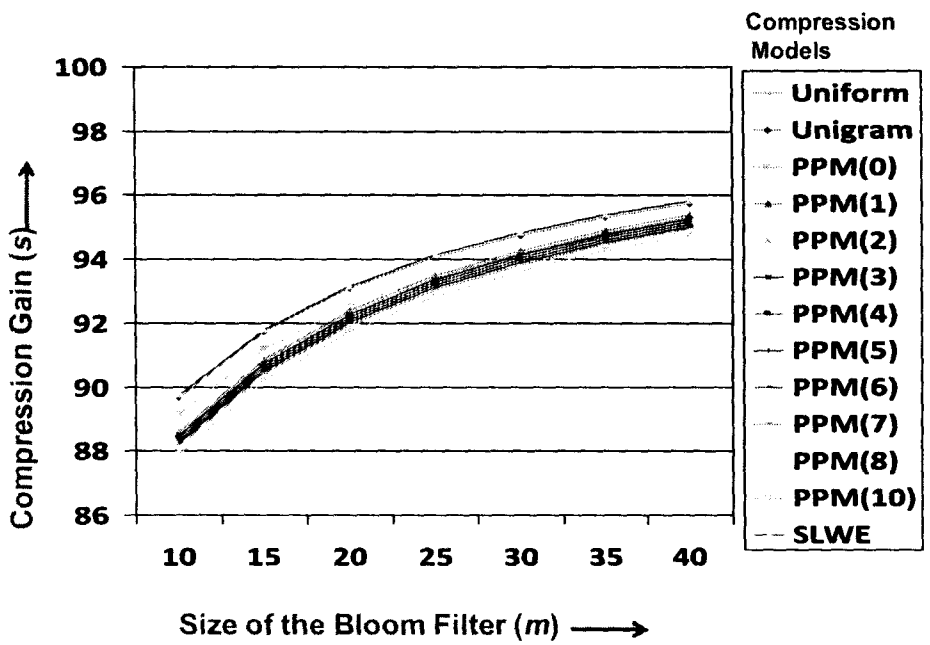

Figure 5-6: Effect of the size of the Bloom filter (m) on the compression gain with respect to the different compression models.

5.3.2.2. Number of Hash Functions (k)

The tests were performed by varying the number of hash functions used to create the Bloom filter. By increasing the number of hash functions, we can test the adaptability of the proposed delta Bloom filter compression methods. Bloom filters of size 140Kb were used. Table 5-7 shows in detail the compression gain for the unigram, PPM and SLWE based delta Bloom filter compression. Uniform means that there was no compression. It

can be noticed that the PPM model-based proposed compression method performs better than the unigram based compression irrespective of the number of hash functions used. The SLWE-based proposed compression method performs even better than the PPM model. For the given size of the Bloom filter with two hash functions, as used by Mitzenmacher [Mitz2002], our proposed SLWE based delta Bloom filter compression method has a compression gain of 93.959 compared to 93.153 of the 0^{th} order unigram-based compression.

Compression Models	Number of Hash functions, k, used								
	2	3	4	5	6	7	8	9	10
Uniform	-0.080	-0.080	-0.080	-0.080	-0.080	-0.080	-0.080	-0.080	-0.080
Unigram	93.153	91.407	90.014	88.830	87.804	87.118	86.449	86.272	85.940
P(0)	92.976	91.247	89.871	88.687	87.672	86.986	86.312	86.140	85.809
P(1)	93.839	91.876	90.312	88.985	87.861	87.124	86.403	86.203	85.883
P(2)	93.759	91.784	90.237	88.968	87.929	87.146	86.341	86.129	85.815
P(3)	93.696	91.716	90.249	88.905	87.781	86.958	86.152	85.958	85.655
P(4)	93.582	91.659	90.094	88.768	87.592	86.815	86.032	85.855	85.546
P(5)	93.502	91.561	89.957	88.647	87.535	86.826	85.998	85.786	85.403
P(6)	93.473	91.401	89.848	88.607	87.426	86.581	85.740	85.512	85.146
P(7)	93.399	91.327	89.786	88.464	87.261	86.415	85.535	85.300	84.929
P(8)	93.330	91.287	89.751	88.373	87.129	86.255	85.363	85.140	84.723
P(10)	93.193	91.144	89.517	88.075	86.786	85.883	85.026	84.786	84.374
SLWE	**94.199**	**92.437**	**91.039**	**89.855**	**88.822**	**88.133**	**87.469**	**87.276**	**86.942**

Table 5-7: Effect of the number of Hash Functions used to construct a Bloom filter on the compression models.

The graph in the Figure 5-7 shows the effect of the number of Hash Functions used to construct a Bloom filter on the compression models. It shows that the number of has functions used is a factor on which all the considered compression models are dependent upon. The graph reflects the trend of the reduction in compression gain with the increase

in the number of hash functions used to construct a Bloom filter. As we can notice, with the increase in the number of hash functions, SLWE-based compression actually achieves better compression in comparison with the other methods in terms of the compression gain although the use of larger number of hash functions will increase the computational complexity.

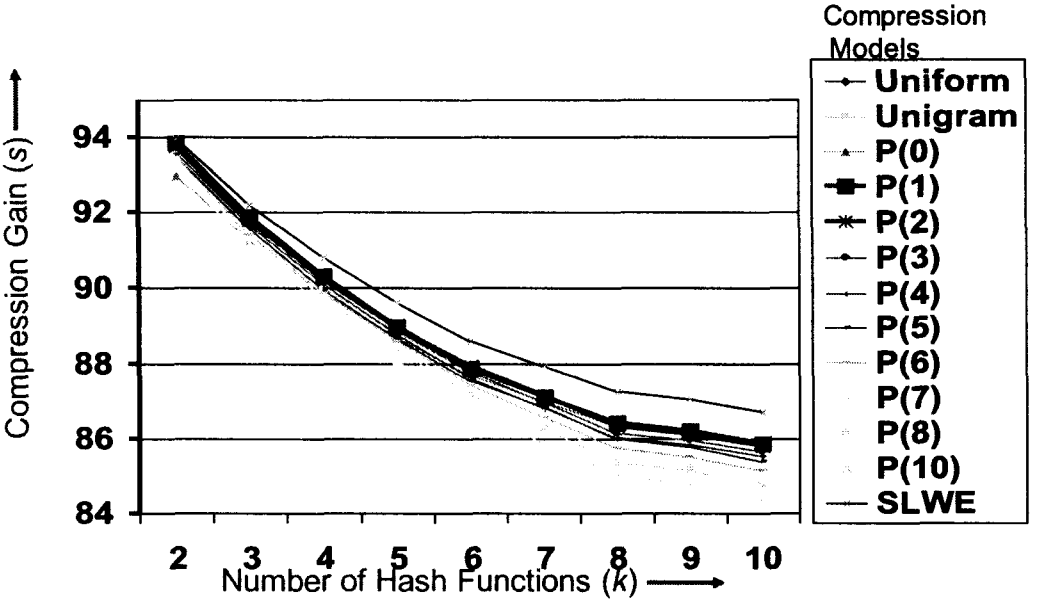

Figure 5-7: Effect of the number of Hash Functions used to construct a Bloom filter on different compression models.

5.3.2.3. Percentage Change Between the Two Bloom filters

The experiments were performed to test the effect of the percentage of change between the two Bloom filters on different compression models. For this comparison, Bloom filters of size 140Kb, created using two hash functions were used. Uniform means there is no compression. Except for the 1% of change, SLWE-based compression leads to better compression gain than the PPM-based compression model and the unigram model, as clearly reflected in Table 5-8. This reflects the adaptability of the proposed compression method for delta Bloom filter compression as though all the considered compression models depend on this factor but the SLWE-based method achieves consistent better compression gain than the unigram or PPM-based methods .

Models	% of change between two Bloom filters, c									
	1	2	3	4	5	6	7	8	9	10
Uniform	-0.080	-0.080	-0.080	-0.080	-0.080	-0.080	-0.080	-0.080	-0.080	-0.080
Unigram	97.186	96.065	95.052	94.074	93.153	92.249	91.385	90.583	89.823	89.017
P(0)	96.980	95.859	94.863	93.891	92.976	92.083	91.225	90.429	89.674	88.868
P(1)	98.215	96.985	95.916	94.852	93.839	92.844	91.860	90.949	90.120	89.228
P(2)	98.187	96.951	95.864	94.772	93.759	92.747	91.780	90.909	90.057	89.160
P(3)	98.164	96.905	95.790	94.715	93.696	92.695	91.700	90.823	90.040	89.125
P(4)	98.147	96.860	95.704	94.617	93.582	92.581	91.649	90.772	89.891	88.942
P(5)	98.124	96.802	95.653	94.543	93.502	92.552	91.523	90.635	89.760	88.788
P(6)	98.095	96.762	95.590	94.486	93.473	92.444	91.403	90.520	89.628	88.662
P(7)	98.061	96.728	95.538	94.440	93.399	92.352	91.317	90.492	89.600	88.639
P(8)	98.044	96.694	95.515	94.406	93.330	92.278	91.254	90.417	89.445	88.462
P(10)	97.998	96.619	95.424	94.240	93.193	92.140	91.151	90.246	89.262	88.239
SLWE	**98.144**	**97.070**	**96.086**	**95.125**	**94.199**	**93.285**	**92.415**	**91.603**	**90.844**	**90.040**

Table 5-8: Effect of the percentage of change between two Bloom filters, c, on different compression models.

The graph in the Figure 5-8 represents the data from Table 5-8 for the effect of the percentage of change between two Bloom filters, c, achieved by the different compression models. It can be noticed that SLWE-based compression consistently performs better than unigram and PPM for delta Bloom filter compression. For very frequent updates to the original Bloom filter, c = 1, unigram, PPM and SLWE based compression models achieve very high compression gain with little difference between them. As the value of c increases, the difference between the compression gain achieved by SLWE-based compression and the other models increases. If an application doesn't require a very frequent data updates, then the selection of SLWE-based delta Bloom filter compression method becomes easier.

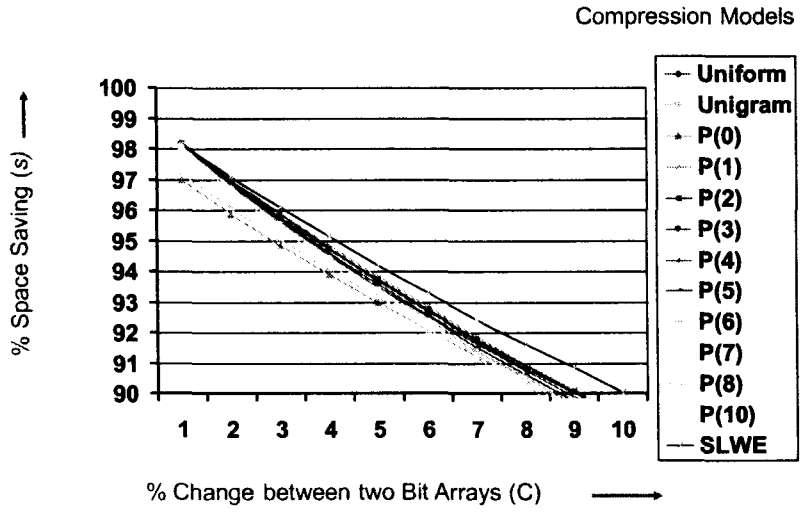

Figure 5-8: Effect of percentage of change between two Bloom filters, *c*, on different compression models.

5.4. Analysis and Discussion

The experiments were performed according to the setup discussed in Section 5.2 of Chapter 5 for considering the dependence and effect of various factors on delta Bloom filter compression. For the first set of experiments, the size of the Bloom filter, *m*, was set to 140,000 bits, which were created using two hash functions, *k*, and the percentage of difference between the two Bloom filters, *c*, was set to 5%. The results of these experiments were very encouraging. The proposed SLWE-based compression had a higher percentage of compression gain, *s*, relative to the benchmark unigram model or the higher-order PPM-based compression. Not only this, the experiments were performed by varying the number of hash functions used, $1 \leq k \leq 10$, it was found that the SLWE based delta Bloom filter compression, again, achieves higher compression gain than the others. This pattern of better compression gain continued even when the experiments were performed with different values of *c*, that is, the percentage of change between the Bloom filters (or the delta Bloom filter).

Encouraged by the experiments results as expected, we went ahead to simulate the real application setup for the proposed delta Bloom filter compression. As discussed in the

fourth chapter, the delta Bloom filter can be used with the Bloom filter-based summary cache for the web proxies. Due to the unavailability a standard benchmark for the real application data, the experiments were performed through simulation. Mitzenmacher has also performed the tests through simulation [Mitz2002].

The size of the Bloom filters was calculated to be used in this scenario. In practice, proxies typically have 8 to 20GB of cache space and an average file size of 8KB. Therefore, the average number of documents can be said to be in the range 1 to 2.5 M documents. The load factor, *lf,* options chosen by Li Fan *et al* [FCAB2000] were 8, 16 or 32, preferably 8 or 16. Based on these findings, the size of the Bloom filter will be:

$$8\text{ Mb} - 20\text{Mb} \quad \text{for} \quad lf = 8,$$
$$16\text{ Mb} - 40\text{Mb for} \quad lf = 16.$$

The percentage of change is in the range $1 \leq c \leq 10$. The optimal number of hash function [MITZ2002] would highly increase the computation cost. Therefore, there is a tradeoff between the false positive rate and the number of hash functions used. Here, four hash functions are used.

In this scenario, also, the second proposed method for delta Bloom filter compression performs better than the first proposed method that uses PPM based compression and the benchmark unigram-based compression, for scalability as well as adaptability, in terms of compression gain.

Chapter 6

CONCLUSION AND FUTURE WORK

As technology evolves, the need for representing and transferring the data concisely and reliably also increases. Bloom filters play a vital role in fulfilling this requirement. However, to fulfill the requirements of the applications which rely on the heavy use of the bandwidth and also the need to access to frequently modified data, the standard Bloom filter use needs enhancement.

6.1. Conclusions

In this thesis, the use of the delta Bloom filter is proposed for such applications and then the delta Bloom filter compression is proposed to increase the benefits. The proposed method for delta Bloom filter compression will bring substantial benefits to the above mentioned applications.

For compressing any data, the selection of the compression tools depends on the characteristics of the data and application for which it is going to be used:

- Streaming versus file.
- Expected patterns and regularities in the data.
- Relative importance of CPU usage, memory usage, channel demands and storage requirements along with other factors.

In this thesis, delta Bloom filter compression is proposed using an advanced estimation technique for arithmetic coding known as stochastic learning-based weak estimation technique that uses a higher-order statistical model and, also along with it, takes into account the variability of the source statistics. Also, delta Bloom filter compression is proposed using higher-order prediction with a partial matching with arithmetic coding. The performance of the proposed compression methods is compared in terms of compression gain to the benchmark compression model and between each other.

Comprehensive experiments were performed to test for the dependence and effect of various factors on the proposed delta Bloom filter compression using SLWE-based and PPM-based compression. The considered factors are the size of the Bloom filter, the number of hash functions used to create Bloom filters, percentage change between two Bloom filters, and parameter λ. The results of the experiments are encouraging in terms of compression gain while also taking scalability and adaptability factors into consideration. The proposed method of the delta Bloom filter compression using stochastic learning-based weak estimation consistently achieves better results in comparison to the other methods.

Due to the unavailability of the standard benchmark real application data, the results are based on simulation. In the future, there is a plan to perform the tests for the proposed delta Bloom filter compression with the data from real applications. A move has already been initiated in this direction and, based on the encouraging results of the simulations, the plan is to present similar results with real application data.

6.2. Future Work

This thesis identifies a need for frequent updates of the data summaries to be transmitted and provides an efficient solution to this problem by reducing the amount of the data being transmitted. In this way, it reduces the bandwidth requirements of a system. The applications involving heavy use of the bandwidth, distributed computing environment for databases or proxy servers, and applications that are sensitive to the access to the data with frequent modifications, all will benefit from the proposed technique. For these applications, there is another aspect to be considered and that is related to security. The next logical step should be a study to combine both these aspects of compression and security. The proposed method provides a solution for large data sets and frequent data transfers. As a future work, a combined solution for large data transfer as well as in encryption can be carried out. For this, the probable areas of study can be the type of hash

functions to be used, combination of compression and encryption algorithms, and trade-off between the combined method and the computational limits of the system.

In the proposed method, cryptographic hash functions are not used. However, we plan to use hash functions such as MD5 and SHA. A future study can be carried out for the use of different types of hash functions to be used with the proposed method.

Also, as a future work, further study and experiments can be performed to learn λ, parameter used for SLWE, while encoding. In this thesis, the range set for parameter λ is 0.9900 to 0.9999 based on previous studies [OR2006]. A set block from the data can be utilized to learn λ and that value of λ can be utilized to learn the probability of the next symbol. Also, further studies can be continued on the use of non-linear stochastic learning-based estimation for the delta Bloom filter compression. Within SLWE approaches there are many learning automata schemes that could be utilized such as nonlinear, continuous, discretized, pursuit learning and estimator algorithms [OR2006]. The study of these algorithms combined with entropy-based coding technique can be done for delta Bloom filter compression. In this thesis, SLWE and PPM are used with arithmetic coding for delta Bloom filter compression. As a future work, these methods can be combined with other encoding schemes such as adaptive Fano coding which could be used for delta Bloom filter compression. Also, Burrows-Wheeler transformation or Block sorting which is discussed in Chapter 3 can be used for delta Bloom filter compression. The algorithm works by applying a reversible transformation to a block of input text. The transformation does not itself compress the data, but re-orders it to make it easy to compress with simple algorithms such as move-to-front encoding. Experiment using block sorting was carried out during thesis. The initial results seem promising; however, more tests are desired and can be carried out in future.

BIBLIOGRAPHY

[AC2005] Artan, N. S., and Chao, H. J., "Multi-packet signature detection using prefix Bloom filters", IEEE, *Global Telecommunications Conference,* 3: 6-28, Nov.-2 Dec., 2005.

[ACweb] http://www.data-compression.info/Algorithms/AC/.

[Bloom1970] Bloom, B. H., "Space/time trade-offs in hash coding with allowable errors", *Communications of ACM,* 13(7):422.426, 1970.

[BM2003] Broder, A., and Mitzenmacher M., "Network Applications of Bloom filters: A Survey", *Internet Math.*, 1(4):485-509, 2003.

[CLL2008] Chen, Y., Lin, I., Lei, C., and Liao, Y., "Broadcast Authentication in Sensor Networks Using Compressed Bloom filters", *Lecture Notes in Computer Science* 5067, pp: 99–111, 2008.

[CW1984] Cleary, J. G. and Witten, I. H., "Data compression using adaptive coding and partial string matching", *IEEE Trans. Communication. COM-32*, pp: 396–402, 1984.

[CYL2006] Chen, L., Ye, S., and Li, X., "Template detection for large scale search engines", In *Proc. of the 2006 ACM symposium on applied computing,* pp: 1094-1098, 2006.

[DKT2003] Dharmapurikar, S., Krishnamurthy, P., and Taylor, D. E., "Longest prefix matching using Bloom filters" , In *Proc. of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (Karlsruhe, Germany,). ACM, New York, NY, pp: 201-212, Aug. 25-29, 2003.

[FCAB2000] Fan, L., Cao, P., Almeida, J., and Broder, A. Z., "Summary cache: a scalable wide area web cache sharing protocol", *IEEE/ACM Transactions on Networking (TON),*8(3):281-293, June, 2000.

[FML2006] Feng, Z., Matt W. M., and Lionel M. N., "The Master Key: A Private Authentication Approach for Pervasive Computing Environments", *In Proc. of the Fourth Annual IEEE International Conference on Pervasive Computing and Communications,* pp: 212-221, March, 2006.

[GNPmc] Gollapudi, S., Najork, M., and Panigrahy, R., "Using Bloom filters to Speed Up HITS-like Ranking Algorithms", *research.microsoft.com.*

[GSQD2004] Guofei, G., Sharif, M., Qin., X., Dagon, D., Lee, W., and Riley, G., "Worm detection, early warning and response based on local victim information", *In proc. Of Computer Security Applications Conference, 20th Annual,* pp: 136-145, 6--10 Dec., 2004.

[HHJ1998] Hankerson, D., Harris, G., and Johnson, P. Jr., "Introduction to Information Theory and Data Compression", CRC Press,1998.

[IMSGI2006] Irmak, U., Mihaylov, S., Suel, T., Ganguly, S., and Izmailov, R, "Efficient *Q*uery Subscription Processing for Prospective Search Engines", In *proc. of the 15th international conference on World Wide Web,* ACM Press, pp:1037-1038, 2006.

[JDT2005] Jain, N. , Dahlin, M. and Tewari, R., "Using Bloom filters to Refine Web Search Results", In the proc. of *8th International Workshop on the Web and Databases,* Baltimore, Maryland, June 16-17, 2005.

[JDT2005] Jain, N., Dahlin, M., and Tewari R., "TAPER: Tiered Approach for Eliminating Redundancy in Replica Synchronization", *USENIX Conference on File and Storage Technologies*, 2005.

[JPEG2009] http://www.jpeg.org/jpeg2000, 2009.

[KZ2006] Kang, L., and Zhenyu Z., "Fast statistical spam filter by approximate classifications", *In proc. of the joint international conference on Measurement and modeling of computer systems,* Saint Malo, France, pp:347.358, 2006.

[LH1987] Lelewer, Debra A.,and Hirschberg, Daniel S., "Data Compression", *ACM Computing Surveys,* 1987.

[LPKS2005] Locasto, M. E., Parekh, J. J., Keromytis, A. D., and Stolfo, S.J., "Towards collaborative security and P2P intrusion detection", *In Proc. of the Sixth Annual IEEE SMC,* pp:333. 339, June 15-17, 2005.

[MertIbm] Mertz, D., "Data Compression Primer: Theory and Strategy of Data Representation", *IBM http://www.ibm.com.*

[MS2009] "BUFFALO: Bloom Filter Forwarding Architecture for Large Organizations / Accountability in Hosted Virtual Networks", *Microsoft Research*, Sept. 2009.

[MNW1998] Moffat, A., Neal, R., and Witten, I. H, "Arithmetic coding revisited" *ACM Transactions on Information Systems*, 16(3), pp:256-294, July 1998.

[Mitz2002] Mitzenmacher, M., "Compressed Bloom filters", *IEEE/ACM Transactions on Networking,* 10(5): 604 . 612, Oct., 2002.

[OR2006] Oommen, B. J., and Rueda, L., "Stochastic learning-based weak estimation of multinomial random variables and its applications to pattern recognition in non-stationary environments", *Pattern Recognition* 39 pp: 328- 341, 2006.

[QLW2007] Qiu, L., Li, Y., and Wu, X., "Preserving privacy in association rule mining with Bloom filters", *Journal of Intelligent Information Systems,* Springer Netherlands, Jan. 27, 2007.

[RCBG2006] Roussev, V., Chen, Y.,Bourg, T.,Golden G., Richard III, "md5bloom: Forensic file system hashing revisited", *Science Direct, Digital Investigation,* 3S(2006), pp: 82-90, 2006.

[RL1979] Rissanen, J. and Langdon, G. G., "Arithmetic coding", *IBM J. Res. Dev. 23*, 2 (Mar.), pp: 149–162, 1979.

[RO2004] Rueda, L., and Oommen, B. J., "On Families of New Adaptive Compression Algorithms Suitable for Time-Varying Source Data", *LNCS* 3261, pp: 234- 244, 2004

[SBB2003] Simpson, M., Barua, R., and Biswas, S., "Analysis of Compression Algorithms for Program Data", August, 2003.

[SQ] Squid Web Proxy Cache, http://www.squid-cache.org. Last access: December 20, 2009.

[Syd2000] Sayood, K., "Intoduction to Data Compression", *Morgan Kaufmann Press,* 2000.

[WTL2008] Wang, H., Tan, C.C., Li, Q., "Snoogle: A Search Engine for the Physical World", IEEE, *In proceedings of INFOCOM 2008 27th Conference on Computer Communications,* pp: 1382-1390, April, 2008.

[Yoon2010] Yoon, M.K., "Aging Bloom Filter with Two Active Buffers for Dynamic Sets", IEEE Transactions on Knowledge and Data Engineering 22(1), pp: 134-138, Jan., 2010.

[YS2006] Yuen, W. H., and Schulzrinne, H., "Improving search efficiency using Bloom filter in partially connected ad-hoc networks", IEEE, In proc. of *Global Telecommunications Conference, San Francisco, CA, USA,* pp:1-5, Nov., 2006.

[ZLSS2008] Zhong, M., Lu, P., Shen, K., and Seiferas, J., "Optimizing Data Popularity Conscious Bloom Filters", ACM, *In Proceedings of the Twenty-Seventh ACM Symposium on Principles of Distributed Computing*, pp: 355-364, Aug. 2008.

[ZL1977] Ziv, J. and Lempel, A.,"A Universal Algorithm for Sequential Data Comression", *IEEE Transactions on Information Theory* 23(3), pp: 337-343, 1977.

[ZL1978] Ziv, J. and Lempel, A., "Compression of Individual Sequences Via Variable-Rate Coding", *IEEE Transactions on Information Theory* 24(5), pp: 530-536, 1978.

VITA AUCTORIS

Priyanka Trivedi was born in Lucknow, India. She received her Bachelor of Engineering degree in Computer Science and Engineering from the Shivaji University, India. Presently, she is completing her Masters' degree in Computer Science from the University of Windsor, ON, Canada and expects to graduate in January 2010.