NETWORK COMPRESSION VIA NETWORK MEMORY: REALIZATION PRINCIPLES AND CODING ALGORITHMS

A Thesis
Presented to
The Academic Faculty

by

Mohsen Sardari

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Electrical and Computer Engineering

Georgia Institute of Technology
December 2013

NETWORK COMPRESSION VIA NETWORK MEMORY: REALIZATION PRINCIPLES AND CODING ALGORITHMS

To my family,

for their love and support.

ACKNOWLEDGEMENTS

I have had the privilege to meet wonderful people in my life's journey who generously provided me with their help and support. I wish to express my gratitude to all of them.

I would like to especially thank my advisor, Dr. Faramarz Fekri. I am grateful to him for giving me the chance to pursue my studies and guiding me through my research. What I have learned from him forms the basis of my ability to face challenging problems throughout my life.

I would also like to express my gratitude to my thesis committee members: Dr. Steven McLaughlin, Dr. Raghupathy Sivakumar, from whom I had the opportunity to learn about wireless networks and further be inspired by his knowledge, personality and exemplar clear-thinking ability, Dr. Jennifer Michaels, and Dr. Howard Weiss with whom I had the chance to take my first graduate course in classical probability. I thank all of my outstanding professors for their support, inspiration, suggestions and encouragement.

Many thanks to my close friend and primary collaborator Ahmad Beirami, the one person who has always been there for me during the ups and downs of my life. It was a great pleasure for me to discuss research and random topics with someone as brilliant as Ahmad. I will miss our afternoon tea sessions!

I would also like to thank my colleagues and friends at CSIP, specially Arash Einolghozati, Nima Torabkhani, Dr. Badri Vellambi, and Jun Zou. My life at Georgia Tech would have not been pleasant without my friend, colleague, and gym-mate Dr. Josep Miquel Jornet whose great personality, sense of humor and top-notch cooking skills made for great memories. His Thanksgiving parties together with Dr. Massimiliano

Pierobon and Dr. Yahia Tachwali made us feel at home, though thousands of miles away from home. I would also like to express my gratitude to my friends, Farshid Ghasemi, Reza Pourabolghasem, Payam Alipour, Alireza Khoshgoftar Monfared, Mehdi Ramezani, Dr. Peyman Kazemian, Rasool Zandvakil, Dr. Saeed Mohammadi, Dr. Mehran Tehrani, Farzad Inanlou, Mehdi Kiani, Arashk Norouzpour, Ali Payani and many more that I cannot name all here, for making my life outside work fulfilling and meaningful. Many thanks to Farshid Ghasemi, Nima Torabkhani and Alireza Khoshgoftar Monfared for their care and support during my defense preparation.

I also greatly appreciate the help and support of Patricia Dixon, Jennifer Lunsford and Cordai Farrar who have always been there to count on. I am certain that there have been many more people than those I have had the joy of acknowledging here and I apologize for not naming all of them.

Most importantly, I am very thankful to my family: my parents, for all the love and support they have always provided, my dear Firoozeh who is the source of my happiness, my sister-in-laws Marjan and Shadab, my brother Hamid, and my sisters Niloofar and Nasim who have always been supportive and encouraging. I dedicate this work to them as a token of my appreciation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The objective of this dissertation is to investigate both the theoretical and practical aspects of redundancy elimination methods in data networks. Redundancy elimination provides a powerful technique to improve the efficiency of network links in the face of redundant data. In this work, the concept of network compression is introduced to address the redundancy elimination problem. Network compression aspires to exploit the statistical correlation in data to better suppress redundancy. In a nutshell, network compression enables memorization of data packets in some nodes in the network. These nodes can learn the statistics of the information source generating the packets which can then be used toward reducing the length of codewords describing the packets emitted by the source. Memory elements facilitate the compression of individual packets using the side-information obtained from memorized data which is called "memory-assisted compression". Network compression improves upon de-duplication methods that only remove duplicate strings from flows.

The first part of the work includes the design and analysis of practical algorithms for memory-assisted compression. These algorithms are designed based on the theoretical foundation proposed in our group by Beirami *et al.* The performance of these algorithms are compared to the existing compression techniques when the algorithms are tested on the real Internet traffic traces. Then, novel clustering techniques are proposed which can identify various information sources and apply the compression accordingly. This approach results in superior performance for memory-assisted compression when the input data comprises sequences generated by various and unrelated information sources.

In the second part of the work the application of memory-assisted compression in wired networks is investigated. In particular, networks with random and power-law graphs are studied. Memory-assisted compression is applied in these graphs and the routing problem for compressed flows is addressed. Furthermore, the network-wide gain of the memorization is defined and its scaling behavior versus the number of memory nodes is characterized. In particular, through our analysis on these graphs, we show that non-vanishing network-wide gain of memorization is obtained even when the number of memory units is a tiny fraction of the total number of nodes in the network.

In the third part of the work the application of memory-assisted compression in wireless networks is studied. For wireless networks, a novel network compression approach via memory-enabled helpers is proposed. Helpers provide side-information that is obtained via overhearing. The performance of network compression in wireless networks is characterized and the following benefits are demonstrated: offloading the wireless gateway, increasing the maximum number of mobile nodes served by the gateway, reducing the average packet delay, and improving the overall throughput in the network. Furthermore, the effect of wireless channel loss on the performance of the network compression scheme is studied. Finally, the performance of memory-assisted compression working in tandem with de-duplication is investigated and simulation results on real data traces from wireless users are provided.

CHAPTER I

INTRODUCTION AND RELATED WORK

A large amount of content is transferred repeatedly across network links in currently existing networks. Several studies have examined real-world network traffic and concluded the presence of considerable amounts of redundancy in the traffic data [80, 7, 8, 87, 68]. From these studies, *redundancy elimination* has emerged as a powerful technique to improve the efficiency of data transfer in data networks.

Currently, the redundancy elimination techniques are mostly based on application-layer content caching. However, several experiments confirm that the caching approaches, which take place at the application layer, do not efficiently leverage the network redundancy which exists mostly at the packet level [87]. Furthermore, caching approaches are incapable of suppressing redundancies that exist across multiple connections. They even lose opportunities for suppressing redundancy within one connection because of issues such as small content size. To address these issues, a few recent studies have considered the deployment of redundancy elimination techniques within the network [6, 87], where the intermediate nodes in the network have been assumed to be capable of storing the previous communication in the network and also data processing. These works demonstrate that redundancy in the data is very high such that simple de-duplication, i.e., removing the repeated segments of the traffic, can give considerable bandwidth savings. However, de-duplication works well only when the redundancy across the packets is so high that a large chunk of the packet is simply the repetition of a previously communicated packet. Nonetheless, the redundancy in the data goes beyond mere repetitions of the previous data chunks and de-duplications is unable to capture statistical dependencies in the data. Furthermore, if the repeated chunks appear with a low frequency such that they do not repeat in the memory window, the repetition goes undetected.

Another related line of work is Content-Centric Networking (CCN) [47, 42]. The CCN advocates segmenting data into individually addressable pieces, and proposes an architecture where individually addressable data segments can be cached in the network. However, there are several fundamental differences between redundancy elimination in this work and the previous research on networking named content. The first difference is that our approach deals with the data itself, as opposed to the content name. As a simple example, two independent servers generating the same content but with different names would still benefit from compression, but not the CCN. The second difference is that CCN has a fixed granularity of a packet, whereas one of the core features of compression algorithms is their flexibility to find redundancy in the data stream with arbitrary granularity. In fact, it is suggested that packet level caching, which most of the current techniques are approximately reduced to, offers negligible benefits for typical Internet traffic [87], due to this predefined fixed granularity.

Motivated by the benefits of redundancy elimination at the packet-level, and to overcome the shortcomings of the previous approached, in this work, redundancy elimination is investigated from an information-theoretic point of view. The foundation of source coding was laid by Shannon in his seminal work on communication theory [77]. The fundamental limit of compression of infinite length sequences for the class of universal schemes is established in the Information theory literature [32]. Entropy is the fundamental limit of compression; sequences generated by a source cannot be compressed with a rate below entropy and uniquely decoded. A compression scheme is called universal if it does not require any prior knowledge about the source statistics. Hence, it is clear that from the practical point of view, the universal family is more interesting than the non-universal one. However, as shown in [12], there is a significant penalty, i.e., gap from the asymptotic limit, when finite length sequences are compressed under a universal scheme. In [13], memory-assisted universal compression was suggested as a potential solution to avoid this penalty. In memory-assisted universal compression, a sequence of length m is obtained from the information-generating source; this sequence is stored at both the encoder and the memory node. This length m sequence, called "memory" sequence, conveys important information about the source statistics. It turns out that memory significantly improves the performance of universal compression for every newly generated sequence from the source. Thus, closing the gap between universal compression performance of finite length and infinite length sequences. This work expands memory-assisted compression in a single link to networks in a framework called "network compression".

Currently, the underlying fabric of the networks consisting of the content sources and server, routers, and clients perform very little, if any, memorization and hence the memory is not utilized for redundancy elimination. The only form of memorization a network performs is application-level caching used by solutions such as web-caches [63, 25], content-distribution networks (CDNs) [81], and peer-to-peer (P2P) applications [86, 67, 31]. The goal of network compression via network memory is to utilized the memory element in the network in order to exploit the statistical correlation in network data to better suppress redundancy. In a nutshell, equipping the network with network memory elements will enable memorization of data traffic as it flows naturally (or by design) through the network. As such, memory enabled nodes can learn the source data statistics which can then be used (as side information) toward reducing the cost of describing the source data statistics in compression.

Network compression is a new paradigm in compression which opens up new research directions beyond network traffic. It requires establishing new fundamental limits and new compression schemes that would take memory into account. This work tackles some of these fundamental questions that will pave the way for realization of network compression. Additionally, for network compression to be practically useful, efficient compression algorithms are presented by which similar contents are identified on the fly and used for the better compression of future similar data. This work is then extended to find achievable network-wide gain of memory deployment in random and Internet-like power-law network graphs. In this model it is assumed that the memory nodes store the previous packets which have passed through them. It is further assumed that the stored packets are utilized in the same way as in memory-assisted compression, which in turn, results in the network flow compression. Finding the achievable gain of compression in the network entails, first finding the optimal location of the memory units in the network, i.e., the memory deployment problem, and then finding the optimal routing algorithm for network compression, given that the memory units' locations are known. The scope of this research is further extended to wireless networks.

While relevant, the network compression problem is different from those addressed by distributed source compression techniques (i.e., the Slepian Wolf problem) that target multiple correlated sources sending information to the same destination [79, 75]. In the Slepian-Wolf, the gains are achievable in the asymptotic regime. Further, the memorization of a sequence that is statistically independent of the sequence to be compressed can result in a gain in memory-assisted compression, whereas in the Slepian-Wolf problem, the gain is due to the bit-by-bit correlation between the two sequences.

Broadly, this work investigates practical and theoretical aspects of the design and analysis of network compression and its realization in wired and wireless networks. The body of work can be divided into three parts. The first part focuses on the design and analysis of practical memory-assisted compression algorithms, whereas the second part concerns with the application of memory-assisted compression in wired networks.

The final part of the work deals with network compression in wireless networks. The outline of the contribution of this dissertation are presented below.

1.1 Redundancy Elimination and Memory-assisted Compression

Since application-level caching cannot capture the packet-level redundancy, development of new content-aware approaches capable of redundancy elimination at the packet and sub-packet levels is necessary. These requirements motivate the redundancy elimination of packets from an information-theoretic point of view. Compression of network packets requires data processing to be performed on individual pieces of data which are of finite sizes. However, traditional universal compression solutions would not perform well over the finite-length sequences, as such techniques can effectively remove redundancy only in the asymptotic regime.

In Chapter 2, a simple parametric source is first considered and two algorithms from dictionary-based and statistical families of universal data compression algorithms are compared. These algorithms are based on Lempel-Ziv (LZ) universal compression [88], and context tree weighting (CTW) universal compression [85, 84] which are adapted for memory-assisted compression. The compression performance of these two algorithms are characterized and it is shown that statistical compression method out performs the dictionary-based method.

Clearly, a simple source cannot model the data traffic. In Chapter 3, the scope of algorithms developed for simple sources are extended to the memory-assisted universal compression when the data sequences are generated by a compound source which is a mixture of parametric sources. To accommodate the compound source, a clustering technique within the memory-assisted compression framework is presented to better utilize the memory by classifying the data sequences from a mixture of simple information sources. Theoretical results on the achievable gains of memory-assisted compression of compound sources with and without clustering as the sequence length and memory sizes vary are obtained. Furthermore, computer simulations are performed to evaluate the performance of memory-assisted compression and validate the benefits of clustering for compression.

The application of clustering to improve compression is a novel concept that is resulted from the idea of network compression via memory. Research in information theory has studied the use of compression for clustering [46, 45, 29] but not other way around. This by itself opens up new exciting research directions both in theory and algorithms.

1.2 Network Compression in Wired Networks

In Chapter 2 and Chapter 3, the benefits of memory-assisted compression were established. It was shown that memorization or learning of past traffic at intermediate nodes provide extra compression gain. This gain comes from the fact that utilizing previous traffic shared between the source and intermediate memory nodes with memory helps to close the gap between the compression performance of universal compression techniques and entropy of each individual sequence. As the next step, the benefits of enabling intermediate nodes in networks with the capability of storing the past communication are explored theoretically. In Chapter 4, memory-assisted compression is applied in Erdős-Rényi random network graphs [33] consisted of a single source and several randomly selected memory units. Analogous to the memory-assisted compression gain for a single link, network-wide gain of compression, or gain of network compression in short, is defined and studied in Erdős-Rényi random network graphs. The gain of network compression depends on the number of memory units; it is discovered that there exists a threshold value for the number of memories deployed in a random graph below which the network-wide gain of memorization vanishes.

In Chapter 6, network compression in Internet-like random power-law network graphs is studied. Similar to Chapter 4, the gain of network compression in Internet-like power-law graphs is characterized. In particular, through analysis on power-law graphs, it is demonstrated that non-vanishing network-wide gain of memorization is obtained even when the number of memory units is a tiny fraction of the total number of nodes in the network. Furthermore, memory placement in the network poses some challenges to traditional shortest path routing algorithms, as the shortest path is not necessarily minimum cost route in networks with memory. Simulation results are presented which validate the results of the analytical study.

In Chapter 5, the routing problem for compressed flows is solved and a modified Dijkstra's algorithm is presented. The last problem studied in Chapter 5 is the memory placement in non-random network graphs. It is shown that optimal memory-placement is not tractable in general network graphs and the challenges involved are demonstrated by deriving the optimal memory placement on line networks.

1.3 Network Compression in Wireless Networks

In Chapter 7 the network compression in wireless networks is investigated. Data traffic in wireless networks is ever increasing. Nevertheless, traces derived from the real-world traffic show significant redundancy at the packet level in the mobile network traffic. This has inspired new solutions to reduce the amount of redundancy present in the packet data in order to manage the explosive traffic. In Chapter 7, a novel approach is proposed to leverage this redundancy for reducing network flows by employing network compression techniques via overhearing memory units deployed as helpers in a wireless network. Each memory-enabled helper overhears the data packets previously sent by the wireless gateway to various mobile clients within its coverage and uses them toward forming a model about the content of the packets from the traffic. The resulting model is then used as side information by the wireless network compression module in a two-part code with asymmetric cost (where the helper-client link is far less costly than the server-client link). It is demonstrated that wireless network compression via overhearing helpers has a three-fold benefit: 1) offloading the wireless gateway and hence increasing the maximum number of mobile nodes the gateway can reliably serve, 2) reducing the average packet delay, and 3) improving the overall throughput in the network. Furthermore, a practical two-part coding algorithm is presented that incorporates the asymmetric cost for the wireless network compression via overhearing helpers. Moreover, the trade-offs between the number of clients, memory-enabled helpers, total throughput and average delay are investigated via extensive simulations. It is demonstrated that the wireless network compression via overhearing helpers significantly improves all of the above performance benchmarks for both UDP and TCP traffics.

In Chapter 8, the impact of packet loss on memory-assisted compression is explored theoretically. In particular, the problem of lossless compression of finite length sequences with mismatched side information at encoder and decoder is considered. The information-theoretic formulation for the problem in the context of universal compression is presented and bounds on the fundamental limit of the compression performance in this setup is derived. Furthermore, in Section 8.4, a sequential code design is presented for memory-assisted compression with mismatched side information.

In Chapter 9, the interplay between memory-assisted compression and de-duplication algorithms is investigated. In particular, we study the de-duplication using Rabin Finger printing [60, 68] working in tandem with memory-assisted compression. The clustering algorithms proposed in Chapter 3 are adapted for working with large real-world data sets gathered from mobile users.

CHAPTER II

MEMORY-ASSISTED COMPRESSION FOR SIMPLE SOURCES

2.1 introduction

In Chapter 2, the concept of memory-assisted compression is introduced and experimental results are presented for memory-assisted compression of simple sources. The results of this chapter provide a foundation for development of the following chapters in this dissertation.

2.1.1 Contributions

In this chapter two compression families, namely, statistical compression family and dictionary-based compression family are introduced and their adaption for memory-assisted compression are presented. Furthermore, the gain of memory-assisted compression is defined and characterized for both compression families [71].

2.2 Setup

Consider an information source node S which generates content to be delivered to a destination (client) node $D \in \mathbf{D}$ connected to S through memory node μ, as shown in Figure 1. Let $x^n = (x_1, ..., x_n)$ be a sequence of length n, where each symbol x_i is from the alphabet $\mathcal{A}$. For example, for a 8-bit alphabet that has 256 symbols, each x_i is a byte. Note also that x^n may be viewed as a packet at the network layer generated by source S. Let $\mathbf{E}[l_n(X^n)]$ denote the expected length resulting from the universal compression of x^n. Further, the client nodes in $\mathbf{D}$ request various sequences from the source over time.

Here, we consider two compression scenarios, as follows:

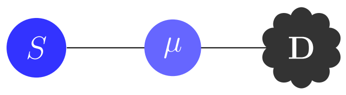


Figure 1: The basic source, memory, and destination configuration. The node **D** represents a set of clients receiving data from S.

1. universal compression of an individual sequence with no memorization (Ucomp), in which a traditional universal compression is applied on the sequence x^n without context memorization, and

2. memory-assisted universal compression of an individual sequence (UcompM), in which the encoder (e.g., server S in Figure 1) and the decoder (e.g., at the intermediate node μ in Figure 1) both have access to a common memory from the same information generating source (to be explained), and they utilize memory for compression of the sequence x^n.

In Ucomp, the intermediate nodes simply forward source packets to the sub-network D. As such, compression takes place in the source and decompression is performed in the destination. Assuming a universal compression at source, $\mathbf{E}[l_n(X^n)]$ would be the length of the compressed sequence, which has to travel within the network from S to the destination D through the intermediate node μ. Since every client in D requests a different sequence x^n over time, the source must encode each sequence x^n independently and route through μ. Now, consider the second scenario in which the intermediate node μ, while serving as an intermediate node for different contents destined for different clients, also constructs a model for the source S. As both source and intermediate nodes are aware of the previous content x^n sent to another client in D, they can leverage this knowledge for the better compression of the traffic sent over the $S - \mu$ portion of the path.

Specifically, assume that previous sequences $x^{m_1}, \ldots, x^{m_L}$ are sent from S to clients $D_1, \ldots D_L$ in the sub-network D via μ. Under UcompM, the node μ constructs a model for the source S by observing the entire length $m = m_1 + \ldots + m_L$ sequence. Note that forming the source model by node μ is not a passive storage of the sequences $x^{m_1}, \ldots, x^{m_L}$. This source model would be extracted differently for different universal compression schemes that we will use as the underlying memory-assisted compression algorithm. UcompM, which utilizes the memorized sequences of total length m, strictly outperforms Ucomp. This benefit, offered by memorization at node μ, would provide savings on the amount of data transferred on the link $S - \mu$ without incurring any penalty except for some linear computation cost at node μ. Please note that the memorization is used in both the encoder (the source) and the decoder (node μ). Thus, source model is available at both S and μ.

Let $\mathbf{E}[l_{n|m}(X^n)]$ be the expected code length for a sequence of length n given a memorized sequence of length m. The fundamental gain of memory-assisted compression $g\,(n, m)$ is defined as

$$g\,(n, m) \triangleq \frac{\mathbf{E}[l_n(X^n)]}{\mathbf{E}[l_{n|m}(X^n)]}.$$

In other words, $g(n, m)$ is the compression gain achieved by UcompM for the universal compression of the sequence x^n over the compression performance that is achieved using the universal compression without memory.

From now on, by memory size we mean the total length m of the observed sequences from the source at the memory unit. To investigate the gain of memorization in the compression of the network flow, we must consider two phases. The first is the memorization phase in which we assume memory units have observed one or multiple sequences of total length m from the source. This phase is realized in actual communication networks by observing the fact that a sufficient number of clients may have previously retrieved different small to moderate length sequences from the server such that, via their routing, each of the memory units has been able to memorize the source and form a model for it. In the second phase, each client may request (a small to moderate length) content from the source. The memory-assisted source coding is performed in the second phase.

Here, perhaps, there is need for some clarifications. First, the memorization and learning from traffic takes place at the network layer because the routers (or the intermediate relays) are observing the packets at the network layer. Therefore, network compression should reside beneath the transport layer and above the network layer, at layer 3.5, as shown in Figure 2. Second, the intermediate node μ must decode and re-encode as the client at destination lacks memory and hence would not be able to decode a packet that is encoded using memory-assisted compression. This implies that if there are multiple routers or relay nodes on the path from the source to the destination, the last memory enabled router (i.e., the one that is closest to the client) must decode the packet using memory-assisted decoding and (potentially) re-encode the result using traditional universal compression before forwarding it to the client. Third, it is reasonable to assume that the client often lacks memory with the source. This is because the client is not connected to the source as often, and hence, even if it has obtained some packets from the source in the past, they may be outdated to carry information about source contents. Whereas, the routers are to observe the source packets much more often and hence have memorized and learned the source contents. Therefore, due to lack of memory at the client, the memory-assisted compression should not be applied end-to-end; from the source all the way to the client.

In the following Section 2.3 and Section 2.4, practical algorithms for memory-assisted compression for simple sources are described, by modifying two well-known compression algorithms, namely the statistical compression methods and the dictionary-based compression method. Furthermore, the achievable gains $g(n, m)$ for real Internet traces are characterized. In particular, HTML and style sheet data was gathered from CNN web server in seven consecutive days. We arbitrarily chose the web server

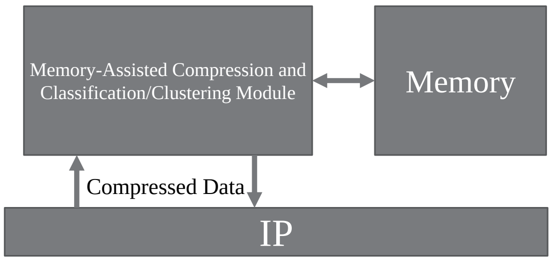

Memory-Assisted Compression and Classification/Clustering Module
Memory
Compressed Data
IP

Figure 2: Network Compression architecture which includes the classification/clustering module.

and similar patterns could be found using data from different web servers. Note that these results are provided as a proof of concept for memory-assisted source coding on real Internet traces.

2.3 Statistical Compression Method

Practical compression algorithms can be divided into two categories: statistical compression method and the dictionary-based compression method. The essence of statistical compression methods is to find an estimate for the statistics of the source based on the currently observed sequence or an external auxiliary sequence. As such, the compression engine follows a two part design, a predictor followed by an arithmetic coder [50], as shown in Fig 42(a). The predictor estimates the statistics of the source and a model is created using the previously seen symbols; based on this model predictions about the probability of the next symbol are issued. In short, the encoding of every new symbol entails:

1. estimating the probability of the symbol based on the model,
2. sending the estimated probability to the arithmetic encoder, which encodes the symbol, and
3. updating the model with the new data. As expected, the decoding process is similar to the encoding.

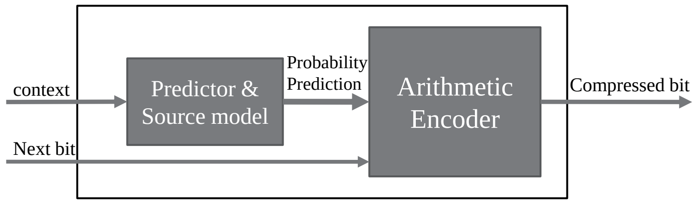


Figure 3: The two-part design of statistical compression algorithms composed of a predictor module and an arithmetic coding module. The predictor maintains a model for the source to use for prediction of probability of the next bit.

The predictor can employ large selection of simple models and combine them to create the source model. A simple and effective predictor can be constructed using tree models; Context Tree Weighting (CTW) algorithm is a well-known example of this approach [85, 84]. CTW is used for experiments of this sections. In CTW, a tree of fixed depth δ is formed to represent the source model; the nodes on the tree correspond to estimates for the statistics of the source. Each bit is compressed according to the previous δ bits called context. Context bits determine a path in the tree that leads to one of the lea nodes. The probability of the next bit is predicted by the information stored in the leaf node. The predicted probability is then sent to a binary arithmetic coder for compression. The tree nodes along the path are then updated using the next bit.

The generalization of the CTW encoding/decoding algorithm for the case of memory-assisted compression is immediate. As previously discussed, in memory-assisted compression, a sequence from the source is available to both the decoder (at μ) and the encoder (at S). This sequence is the concatenation of all the packets sent from S to μ in Figure 1. Therefore, using this sequence, a context tree can be constructed that will be further updated in the compression process. Note that the source and memory node should always keep the context tree synchronized with each other. In practical settings, a simple acknowledgment mechanism suffices for the context synchronization.

2.4 Dictionary-based Compression Method

Unlike the statistical compression methods that rely on the estimation of the source statistical parameters, dictionary-based compression methods select sequences of symbols and encode each sequence using a dictionary of sequences that is generally constructed using the previously compressed symbols. The dictionary may be static or dynamic (adaptive). The former does not allow deletion of symbols from the dictionary, whereas the latter holds symbols previously found in the input stream, allowing for additions and deletions of symbols as new input is being read.

One of the most efficient dictionary-based methods, that is investigate in this work, is the LZ77 algorithm [88]. The principle of LZ77 is to use part of the previously-seen input stream as the dictionary. The encoder maintains a window to the input stream and shifts the input in that window from right to left as strings of symbols are being encoded.

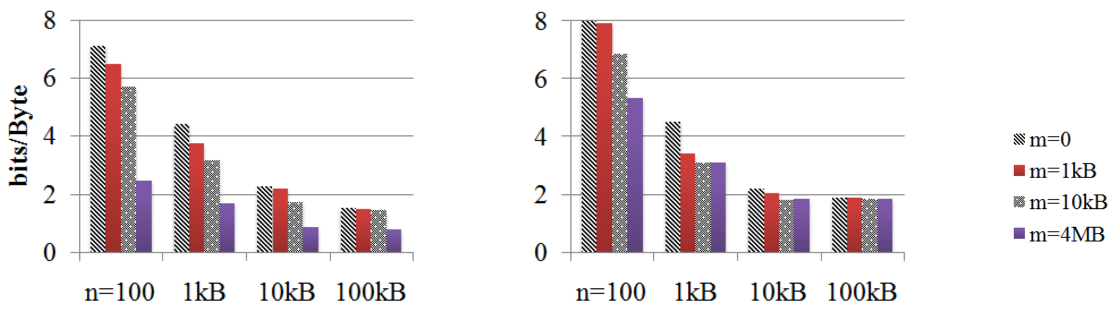
The implementation of the memory-assisted LZ77 in this work is based on the open-source DEFLATE algorithm. A sequence of length m is assumed to be available at both the encoder and the decoder. The previously seen sequence is then used as the common dictionary. The new data to be compressed, is appended to the end of the dictionary at the source and fed to the LZ77 encoder. The output is sent to the decoder. Similarly, the decoder can reconstruct the intended stream by appending the transmitted symbols to the end of the dictionary and perform the LZ77 decoding algorithm.

2.5 Simulation Results

The compression performance of the memory-assisted compression algorithms described in Section 2.3 and Section 2.4 are depicted in Figure 4. We have implemented the memory-assisted CTW and quantified the achievable gains using real Internet traffic data, as shown in Figure 4(a). As expected, the size of the compressed sequence decreases as memory size m increases. For example, for a data sequence of length $n = 100$ Bytes, without memory, the compressed sequence has a length of ≈ 87 Bytes, while using a memory of size $m = 4$MB, this data sequences can be compressed to 31 Bytes; almost 3 times smaller. Our simulation results for memory-assisted LZ77 with a window size of 32kB (all seizes are reported in Bytes) and various dictionary sizes are shown in Figure 4(b).

The actual gain of memory-assisted compression g for memory size 4MB is depicted in Figure 34. Our results suggest that the memory-assisted statistical compression method outperforms the dictionary-based method in both the absolute size of the compressed output and also the gain of memory, i.e., the gain achieved on top of the gain of conventional compression, by utilizing memory.

Both dictionary-based and statistical compressors can achieve the entropy limit for very large input sequences. However, they perform poorly for short to moderate length sequences. Therefore, the main advantage of our memory-assisted compression is that it will overcame this limitation by exploiting the available memory.



(a) Memory-assisted CTW (b) Memory-assisted LZ

Figure 4: The compression ratio (bits/Byte) achieved by memory-assisted algorithms.

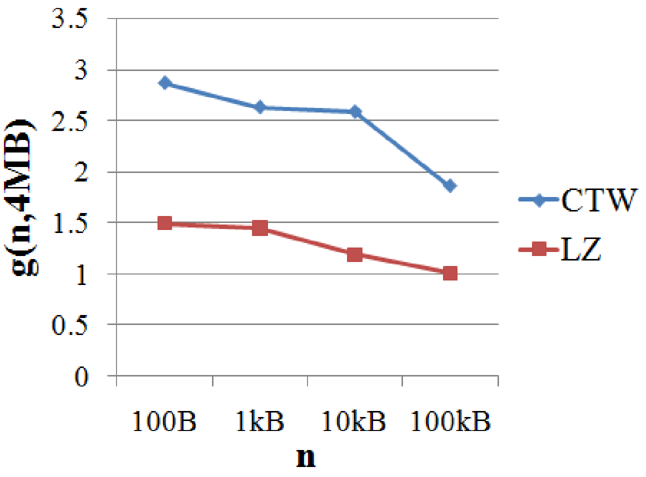


Figure 5: The gain g of memory-assisted compression over traditional compression (Ucomp), for memory size of 4MB for CTW and LZ compression algorithms. This gain is achieved by utilizing memory on top of the performance of the conventional compression.

CHAPTER III

CLUSTERING FOR MEMORY-ASSISTED COMPRESSION OF COMPOUND SOURCES

3.1 Introduction

The algorithms for memory-assisted compression presented in Section 2.3 and Section 2.4 are beneficial for compression of sequences generated by a single information source. However, a single source cannot model the data traffic. Hence, a more complete model for the source that is called “compound source model” is considered. Compound source model is a mixture of simple information sources. The choice of compound source model is motivated by the observation that in practice packets in the network are generated by various applications; every application can be treated as an information source. Therefore, in a compound source model for a content server at the network, the sequences (packets) in the memory are from various information sources. This raises the question that whether a naive application of memory-assisted compression would suffice to achieve the expected memorization gain in memory-assisted compression. The theoretical results in Section 3.4 demonstrate that the crude formation of the context by a memory element from the previously observed packets under certain conditions asymptotically worsen the compression performance. Hence, we consider clustering technique within the memory-assisted compression framework to better utilize the memory by classifying the data packets from a mixture of sources.

3.1.1 Contributions

In Chapter 3 a clustering algorithm is presented which aims at utilizing the data in the memory to better compress a new sequence from the compound source. The main idea of the clustering is to group packets (in the memory) that can be compressed well together. The hypothesis is that the packets in the same cluster would share similar statistical properties and hence improve the compression performance. This hypothesis is examined by extensive simulations on real-world data. A newly generated packet by the compound source is first classified into one of the clusters and then the set of packets in the selected cluster is used as the memory for the compression of the new packet.

In short, to better model a real content-generator server, it is desirable to study a new model in which packets are from a compound (mixture) of several information sources. This new model will influence both the theoretical analysis as well as the algorithms developed for memory-assisted compression. In this chapter, the related issues are addressed and a joint memorization and *clustering* technique for compression is proposed that is suitable for a compound source. The questioned to be answered in this chapter are:

1. How much performance improvement should we expect from the joint memorization and clustering versus the memorization without clustering?

2. How should the clustering scheme be realized to achieve good performance from compression with the joint memorization and clustering?

3. Given a set of clusters, how classification of an incoming new sequence into one of the clusters in the memory is performed such that the performance of memory-assisted compression is maximized?

The results in this chapter were presented in [73, 15]. The theoretical results in Section 3.4 are developed by my colleague, Ahmad Beirami, and are presented for completeness. As such, the proofs are omitted from this dissertation.

3.2 Background

The goal of the clustering is to group the sequences in the memory such that the total length of all the encoded sequences is minimized. Therefore, it is natural to adapt a clustering algorithm, among the many, that has the codelength minimization as its principle criterion. We employ a Minimum Description Length (MDL) approach suggested by [45]. The MDL approach is particularly interesting since it does not need any prior distribution; it only uses the data at hand. The MDL model selection approach is based on finding shortest description length of a given sequence relative to a model class. In other words, MDL clustering tries to group sequences in the memory together in such a way that the resulting total of all the codelengths in the clusters is minimized, which is aligned with our expectation of the clustering algorithm. We should point that the ultimate goal of memory-assisted compression is to utilize the memory in order to minimize the codelength of the "new sequence" using clustered memory. The simulation results in this chapter demonstrate that for cases where the length of memory is larger than the length of the new sequence, the MDL clustering demonstrates a very good performance close to that of compression with source-defined clustering of the memory.

The MDL clustering and classification problem above involves finding the codelength of a set of sequences when grouped and then compressed together. As the MDL relies on the codelength minimization, the complexity of the codelength computation determines the complexity of the algorithm. Therefore, finding efficient ways to find the minimum code length of a set of sequences (or an approximation thereof) to perform the classification and clustering is core to the problems studied in this chapter. Furthermore, the number of simple sources in the compound source can be unknown a priori. In the theoretical development, a reduced form of the problem is considered where it is assumed that the number of information sources in the compound source is known and the number of clusters is chosen to be equal to the number of information sources.

A mixture of sufficient number of parametric sources is expected to model the complex nature of the content generator. Let $\mathcal{A}$ be a finite alphabet and let the parametric source be defined using a d-dimensional parameter vector $\theta = (\theta_1, ..., \theta_d)$, where d denotes the number of the source parameters. For example, if the alphabet size is $||\mathcal{A}|| = 256$ (byte), for a first-order Markov source the number of source parameters is 256×255 which is equal to the number of independent transition probabilities. Denote μ_θ as the probability measure defined by the parameter vector θ on sequences (packets) of length n. We also use the notation μ_θ to refer to the parametric source itself. We assume that the d parameters are unknown.

It is assumed that both the encoder and the decoder have access to a common memory of the previous T packets (each of size n) from the compound source. Let $m = nT$ denote the total memory length. Further, denote $\mathbf{y} = \{y^n(j)\}_{j=1}^{T}$ as the previous T packets shared between M_1 and M_2. We may view $\mathbf{y}$ as the concatenation of previous T packets as well. Note that each of the packets $y^n(j)$ might be from a different source in the compound source. Let $\mathbf{p} = (p_1, ..., p_\mathcal{K})$, where $\sum_{i=1}^{\mathcal{K}} p_i = 1$, as the probability distribution according to which the information sources in the compound source (consisted of $\mathcal{K}$ parametric sources) are selected for packet generation, i.e., the source i is picked with probability p_i. Let the random variable Z_j denote the index of the source that has generated the packet $y^n(j)$, and hence, Z_j follows the distribution $\mathbf{p}$ over $[\mathcal{K}]$. Therefore, at time step j, packet $y^n(j)$ is generated using the parameter vector $\theta^{(Z_j)}$. Further, denote $\mathbf{Z}$ as the vector $\mathbf{Z} = (Z_1, ..., Z_T)$. We wish to compress the packet x^n with source index Z_{T+1}, when both the encoder and the decoder have access to a realization $\mathbf{y}$ of the random vector $\mathbf{Y}$. This setup, although very generic, can occur in many applications. As the most basic example, consider the communication scenario in Figure 6. The presence of the shared memory $\mathbf{y}$ at M_1 and M_2 can be used by the encoder at M_1 to compress (via memory-assisted source coding) the packet x^n which is requested by client C. The compression can reduce the transmission cost on the $M_1 - M_2$ link while being transparent to the client, i.e., M_2 decodes the memory-assisted source code and then applies conventional universal compression to x^n and transmits to C.

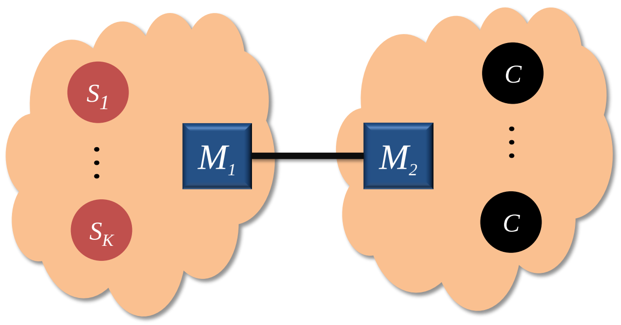


Figure 6: The basic memory-assisted compression scenario between two memory elements M_1 and M_2 in the network. The compound source (i.e., the content server) is shown as a set of multiple simple sources $S_1, \ldots, S_\mathcal{K}$ on the left.

Let $H_n(\theta)$ be the source entropy given θ, i.e.,

$$H_n(\theta) = \mathbf{E}\left[\log\left(\frac{1}{\mu_\theta(X^n)}\right)\right] = \sum_{x^n} \mu_\theta(x^n) \log\left(\frac{1}{\mu_\theta(x^n)}\right). \tag{1}$$

In this chapter $\log(\cdot)$ always denotes the logarithm at base 2. Let $c_n : \mathcal{A}^n \to \{0,1\}^*$ be an injective mapping from the set $\mathcal{A}^n$ of the sequences of length n over $\mathcal{A}$ to the set $\{0,1\}^*$ of binary sequences. Denote $R_n(l_n, \theta)$ as the expected redundancy of the code with length function $l_n(\cdot)$ which assigns a codeword length to x^n, defined as

$$R_n(l_n, \theta) = \mathbf{E}[l_n(X^n)] - H_n(\theta). \tag{2}$$

Note that the expected redundancy is a measure of how close we can compress the packet from the parametric source with parameter vector θ to the fundamental limit given by $H_n(\theta)$.

3.3 *Packet Coding Strategies at Memory Elements*

In order to investigate whether or not memorization provides compression benefit for the compound source, the following three schemes are compared:

- Ucomp (Universal compression), in which a simple universal compression is applied on the packet x^n without regard to the memorized packets $\mathbf{y}$.
- UcompM (Universal compression with naive context memorization), in which the encoder at M_1 and the decoder at M_2 both have access to the memorized sequence $\mathbf{y}$ from the compound source, and they use $\mathbf{y}$ without regard to the index Z for the *naive* learning of the source statistics in the compression of the packet x^n.
- UcompCM (Universal compression with source-defined clustering of the memory), in which the encoder and the decoder both have access to the clustered memory, i.e. the memory sequence $\mathbf{y}$ and the index $\mathbf{Z}$ of the source of memorized packets, from the compound source and use it toward the compression of the sequence. Specifically, the sequence is first classified to one of the clusters in the memory. Then, it is compressed using the sequences memorized in the respective cluster (shared between the encoder and the decoder).

In the theoretical analysis, we use the average redundancy for the quantification of the performance of the different coding strategies. Note that the redundancy rate and the compression rate are interrelated since their difference is the constant entropy rate as can be seen in (2).

Denote $\bar{R}_{\text{Ucomp}}(n)$ as the average minimax redundancy of the Ucomp coding strategy, given by

$$\bar{R}_{\text{Ucomp}}(n) \triangleq \inf_{l_n} \sup_{\theta} R_n(l_n, \theta). \tag{3}$$

In UcompM, let $l_{n|m}$ be the naive memory-assisted length function with a memory sequence of length m. Let $R_n(l_{n|m}, \theta)$ be the expected redundancy of encoding a packet of length n from the source μ_θ using the length function $l_{n|m}$. Further, let $\bar{R}_{\text{UcompM}}(n, m)$ denote the corresponding average minimax redundancy, i.e.,

$$\bar{R}_{\text{UcompM}}(n, m) \triangleq \inf_{l_{n|m}} \sup_{\theta} R_n(l_{n|m}, \theta). \tag{4}$$

In UcompCM, let $l_{n|m,\mathbf{Z}}$ denote the length function for the universal compression of a packet of length n with memorized sequences $\mathbf{y}$, where the vector $\mathbf{Z}$ of the source indices is known. Denote $R_n(l_{n|m,\mathbf{Z}}, \theta)$ as the expected redundancy of encoding a packet x^n of length n using the length function $l_{n|m,\mathbf{Z}}$. Denote $\bar{R}_{\text{UcompCM}}(n, m)$ as the expected minimax redundancy given by

$$\bar{R}_{\text{UcompCM}}(n, m) \triangleq \inf_{l_{n|m,\mathbf{Z}}} \sup_{\theta} R_n(l_{n|m,\mathbf{Z}}, \theta). \tag{5}$$

The following is a trivial bound on the performance of UcompCM.

Remark The average minimax redundancy of UcompCM is smaller than that of Ucomp, i.e.,

$$\bar{R}_{\text{UcompCM}}(n, m) \leq \bar{R}_{\text{Ucomp}}(n). \tag{6}$$

Equation (6) simply states that the context memorization with source defined clustering improves the performance of the universal compression. We stress again that the saving of memory-assisted compression in terms of flow reduction is obtained in the M_1-M_2 link in Figure 6.

3.4 Theoretical Analysis

In this section, we provide discussion on the performance of the different packet coding strategies introduced in the previous section by the analysis of the average minimax redundancy.

The performance of traditional universal compression has been extensively studied in the literature (cf. [12] and the references therein). In the case of Ucomp coding strategy, Clarke and Barron derived the expected minimax redundancy $\bar{R}_{\text{Ucomp}}(n)$ for memoryless sources [30], which was later generalized by Atteson for Markov sources, as the following [9]:

Theorem 3.4.1 *The average minimax redundancy of Ucomp coding strategy is given by*

$$\bar{R}_{Ucomp}(n) = \frac{d}{2}\log\left(\frac{n}{2\pi e}\right) + \log\int |\mathcal{I}_n(\theta)|^{\frac{1}{2}} d\theta + O\left(\frac{1}{n}\right), \tag{7}$$

where $\mathcal{I}_n(\theta)$ is the Fisher information matrix.

Theorem 3.4.1 gives the average minimax redundancy for the Ucomp coding strategy. According to this theorem, the performance of universal compression on finite-length packets, with size similar to IP packets, is fundamentally limited by the inevitable compression overhead (redundancy) imposed by the universal compression.

For the simple case of a single parametric source, Beirami *et al* showed in [73, 15] that the average minimax redundancy of UcompM is given by

$$\bar{R}_{\text{UcompM}}(n, m) = \frac{d}{2}\log\left(1 + \frac{n}{m}\right) + O\left(\frac{1}{n}\right). \tag{8}$$

which also theoretically quantifies our previous results in Chapter 2. Comparing (7) and (8), we observe that the second term in (7) can be eliminated by using memory. This term, in fact, has a large contribution to redundancy of finite-length universal compression. Therefore, when the memory size is large enough, i.e., a sufficient number of packets from previous communication have been stored, the overhead (redundancy) of universal compression due to finite-length constraints becomes subtle.

In [15], Beirami *et al* studied the minimax redundancy of universal compression of sequences from compound parametric sources. It was shown that when the number of sources is more than one, the average minimax redundancy of UcompM coding strategy as $m \to \infty$, is given by $\bar{R}_{\text{UcompM}} = \Theta(n)$.[1] This implies that the naive memorization of the previous packets using UcompM without regard to which source parameter has indeed generated the packet would not suffice to achieve the memorization gain. In fact, the redundancy of UcompM is worse than the redundancy of Ucomp, for large n. This demonstrates that UcompM (without clustering) performs worse than Ucomp when more than one information source is employed. As such, the clustering of the memorized packets is necessary for effective memory-assisted compression in scenarios where the source is a mixture of several parametric sources. In [73], the compression performance in an ideal case where we know the source of each packet is investigated.

In short, the theoretical results presented in Section 3.2 demonstrate that if sufficient memory of the past is present at the memory element, the overhead (redundancy) due to finite-length universal compression may be eliminated via joint memorization and clustering which in turn result in improved compression performance. This result motivates the investigation of memorization and clustering approach in real-world scenarios. In Section 3.5, we will further relax the source-defined clustering assumptions and study the impact of clustering in practice.

3.5 Hierarchical Clustering

In Section 3.5, we try to answer the main question in the memory-assisted compression setup we introduced: "How do we utilize the available memory to better compress a packet generated by a real-world content server?" In Section 3.4, it was shown theoretically that clustering is necessary to effectively utilize the memory in the proposed memory-assisted compression of traffic packets. Within this framework, we identify two interrelated problems: 1) How do we perform clustering of memorized packets

[1] $f(n) = \Theta(g(n))$ if and only if $f(n) = O(g(n))$ and $g(n) = O(f(n))$.

to improve the memory-assisted compression in real-world traffic? 2) Given a set of clustered packets, how do we classify an incoming new packet (to be encoded) into one of the clusters in the memory using which the performance of memory-assisted compression is optimized?

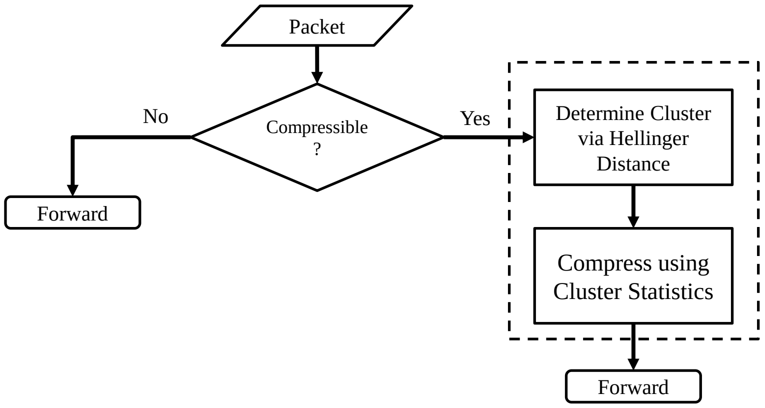
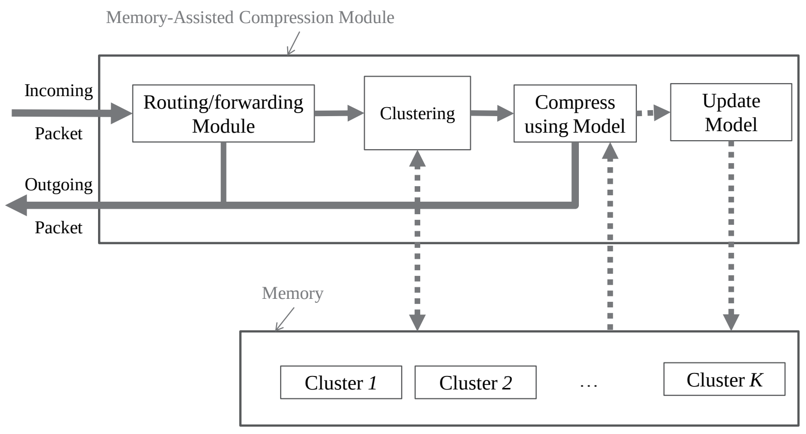

Packet
Compressible ?
No
Yes
Forward
Determine Cluster via Hellinger Distance
Compress using Cluster Statistics
Forward

Figure 7: Network packet compression flowchart. The modules in the dashed box are the components of the K-means clustering using Hellinger distance.

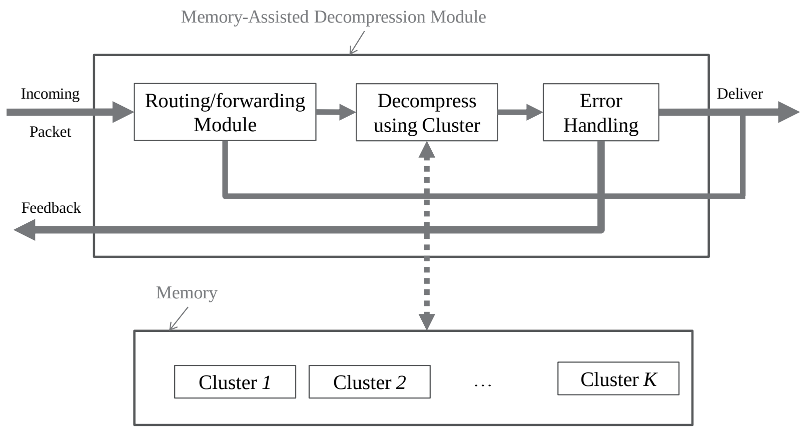
The goal of the clustering is to group the packets in the memory such that the total length of all the encoded packets is minimized. To do so, in the sequel, we describe a hierarchical clustering algorithm that proves to be useful for compression. The proposed hierarchy for the content-aware joint memorization and clustering for network packet compression is shown in Figure 7. As shown, we first identify whether or not an incoming packet is compressible. If the packet is determined incompressible, it is neither compressed nor stored in the memory. On the other hand, if a packet is determined compressible, it is passed to the clustering unit which operates based on the Hellinger distance metric. Figure 8 depicts the details of the compression and decompression modules of the memory-assisted compression with clustering.

3.5.1 Compressibility Determination

We perform the compressibility determination based on the empirical entropy of the data packet. Figure 9 demonstrates the distribution of the empirical entropy of 1000



(a) Compression



(b) Decompression

Figure 8: The compression and decompression modules. The routing/forwarding module handles compressibility determination. The clustering module creates source models and stores them in the memory.

packets downloaded from CNN and Apple websites. The choice of the web servers is arbitrary and similar patterns could be obtained using different websites. As can be seen, using this diagram, the packets may be divided into two categories. One category contains packets with very high entropy rate (close to 8 bits per byte) and hence these packets are incompressible.[2] The other category contains packets whose

[2]Note that exactly 8 bits are required to represent each data byte when no compression is in effect.

empirical entropy rate is estimated to be much less than 8, and hence, these packets are compressible. Therefore, as the first step the packets are partitioned into compressible and incompressible.

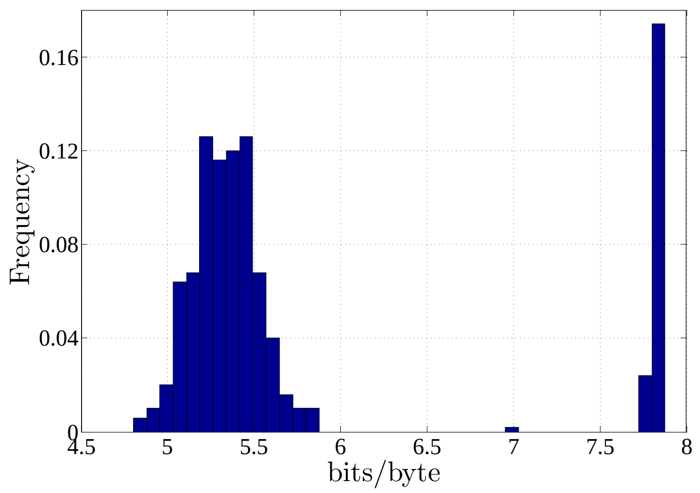


Figure 9: The distribution of the empirical entropy of the packets in the trace under study.

Note that the compressibility determination is very fast and only requires the calculation of the empirical entropy, which is linear with the data packet size. If a data packet is determined to be compressible, it is kept in the memory for memory-assisted compression. On the other hand, whenever a data packet is determined to be incompressible, it is not stored in the memory and simply forwarded without compression. After the partitioning step, the packets in the resulting memory are all compressible. Then, we will perform a clustering of the resulting memory based on the Hellinger distance metric between the packets.

3.5.2 Clustering Using Hellinger Distance Metric

The Hellinger distance metric is a metric to quantify the similarity/difference between two probability distributions (cf.[24]). For two probability distributions $p(x)$ and $q(x)$

on symbols x_i from alphabet $\mathcal{A}$, the Hellinger distance is defined by

$$d_H(p,q) = \frac{1}{2}\sqrt{\sum_{x_i \in \mathcal{A}} \left(\sqrt{p(x_i)} - \sqrt{q(x_i)}\right)^2}. \tag{9}$$

In our setup, we calculate the Hellinger distance of two packets using their empirical distribution of symbols for each individual packet. Recall that a packet $x^n \in \mathcal{A}^n$ is a vector of n symbols (a byte here) $x_i \in \mathcal{A}$.

3.5.2.1 Clustering

We describe as to how we cluster the packets in the memory based on Hellinger distance. A good clustering is such that it allows efficient compression of the whole data set. Equivalently, the packets that are clustered together should share similar statistical properties in order to be compressed well together. Suppose the total number of clusters is given by $\mathcal{K}$, and each packet in the memory (after partitioning step in Figure 7) needs to be assigned to one of the clusters. We use the binary indicator c_t^j to denote the cluster assignment for the $y^n(t)$ (the t-th packet) in the memory, where $c_t^j = 1$ if $y^n(t)$ is assigned to cluster $j \in [\mathcal{K}]$, otherwise $c_t^j = 0$. Then the objective function for clustering is given by

$$J = \sum_{t=1}^{T} \sum_{j=1}^{\mathcal{K}} c_t^j d_H(q_t, u_j), \tag{10}$$

where q_t is the distribution on the symbols obtained from the packet $y^n(t)$ and u_j is the probability distribution vector on the symbols associated with the packets in cluster j. The goal of the clustering algorithm is to find the assignment c_t^j for $j \in [\mathcal{K}]$ and $t \in [T]$ such that J is minimized.

The problem setup suggests that the K-means clustering algorithm [18] is very suitable for our purpose. It is an iterative algorithm which consists of two steps for successive optimization of c_t^j (and hence u_j). Given cluster center u_j, the optimal

c_t^j can be easily determined by assigning the packet $y^n(t)$ to the closest cluster with minimum Hellinger distance $d_H(q_t, u_j)$. Then, we fix c_t^j and update u_j. The clustering algorithm is summarized in Algorithm 1.

Algorithm 1 K-means clustering using Hellinger distance

```
  Obtain the symbol distribution $q_t$ of each packet
  Select $\mathcal{K}$ packets $\{y^n(t_1), y^n(t_2), \dots, y^n(t_{\mathcal{K}})\}$ by picking one packet out of every $\lfloor \frac{T}{\mathcal{K}} \rfloor$ packets
  Generate initial cluster vector $u_j$ using $q_{t_j}$ of the selected $\mathcal{K}$ packets
  repeat
    for $t \in [T]$ do
      $c_t^j = \begin{cases} 1 & \text{if} \quad j = \text{argmin}_i \, d_H(q_t, u_i) \\ 0 & \text{otherwise} \end{cases}$
    end for
    for $1 \leq j \leq \mathcal{K}$ do
      $u_j = \frac{\sum_{t=1}^{T} c_t^j q_t}{\sum_{t=1}^{T} c_t^j}$
    end for
  until convergence or maximum number of iterations
```

3.5.2.2 Classification

Once the clustering of memory is performed, to compress a new packet x^n, we must first decide which cluster must be used as a side information to compress x^n. Therefore, we must classify the packet x^n by assigning it to a proper cluster. The classification algorithm is as follows. Let c be the cluster label of x^n to be determined. We compute Hellinger distance between the symbol distribution q of x^n and the cluster u_j. Then x^n is assigned to the closest cluster as follows

$$c = \underset{1 \leq j \leq \mathcal{K}}{\text{argmin}} \, d_H(q, u_j). \tag{11}$$

3.5.3 Simulation Results

The simulation results in Section 3.5.3 are divided into two parts. The first part is designed to demonstrate the importance of the clustering and also verify the results in Section 3.4. We consider $\mathcal{K}$ first-order Markov sources with alphabet size of 256 and source entropy $H_n(\theta)/n = 1$ bit per byte. The length of the sequences generated by the source are fixed to be equal to n. The number of sources in the mixture are $\mathcal{K} = 10$ and every sequence has equal chance to be from one of these sources. Further, the compression is performed using CTW algorithm and averaged over multiple runs of the experiment.

Figure 10 depicts memory-assisted compression with source-defined clustering, denoted by g_{cm}, for various memory sizes m. Joint memorization and clustering achieves up to 6-fold improvement over the traditional universal compression, Ucomp.

Figure 11 depicts the experimental g_{cm} using CTW, and the experimental gain of naive memory-assisted compression, denoted by g_{m}, for memory $m = 10$MB. Note that g_{m} is the memorization gain (which is the performance benefit of UcompM) over Ucomp when both schemes are used for the compression of a compound source without any clustering technique involved. As expected, with no clustering, the memory-assisted compression may result in a worse compression rate than compression with no memory validating our result in Section 3.4.

The second part of simulations are designed to demonstrate the effectiveness of the content-aware joint memorization and clustering proposed in this work through computer simulations. The proposed algorithm is applied to real-world packets captured from CNN and Apple websites. To capture the packet, we used *wget* and *wireshark* [3] open-source packet analyser together and stored the IP packets. We captured 8100 data packets from each website for our experiment. All packets have the same size of 1434 bytes. We randomly select 100 packets from each website as test packets to measure the compression performance. The remaining 16000 packets are then used to construct memory and the clustering algorithm is performed on these packets.

Figure 12 demonstrates the probability distribution associated with each of the obtained clusters after the convergence of the clustering algorithm. As can be seen,

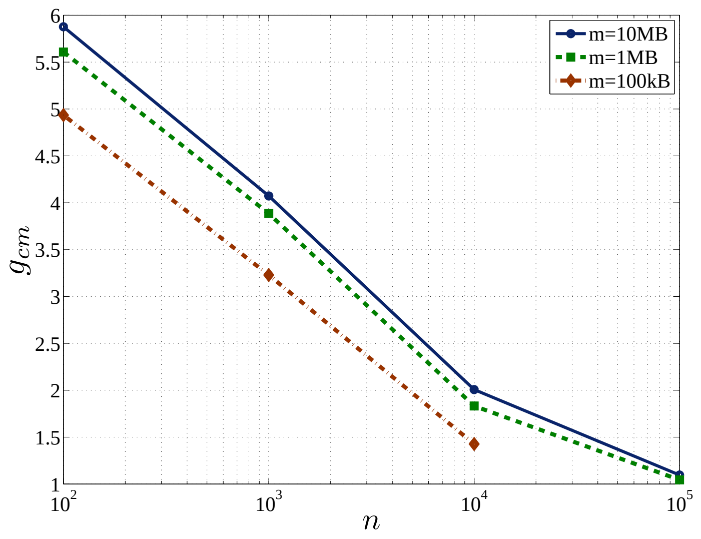


Figure 10: The gain g_{cm} of memory-assisted compression with source-defined clustering.

clusters have significant amount of English alphabet as well as special characters that are attributed to either text or scripts.

In order to illustrate the importance of clustering for efficient use of memory, we have evaluated the compression rate for the following cases after the entropy classification is performed. 1) Ucomp (Universal compression), 2) UcompM (Universal compression with naive memorization), 3) UcompCM (Universal compression with memorization and source-defined clustering), 4) UcompH (Universal compression with memorization and K-means clustering using Hellinger distance).

The simulation results are shown in Figure 13. The compression rate on the compressible packets is plotted for the different packet coding strategies. As can be seen, our proposed clustering scheme based on the Hellinger distance metric achieves superior performance over the traditional universal compression. Note that these results do not demonstrate the impact of the incompressible packets. This is due to the fact that the number of incompressible packets may vary in different websites as some traffic traces contain more images and some contain more text/scripts. However,

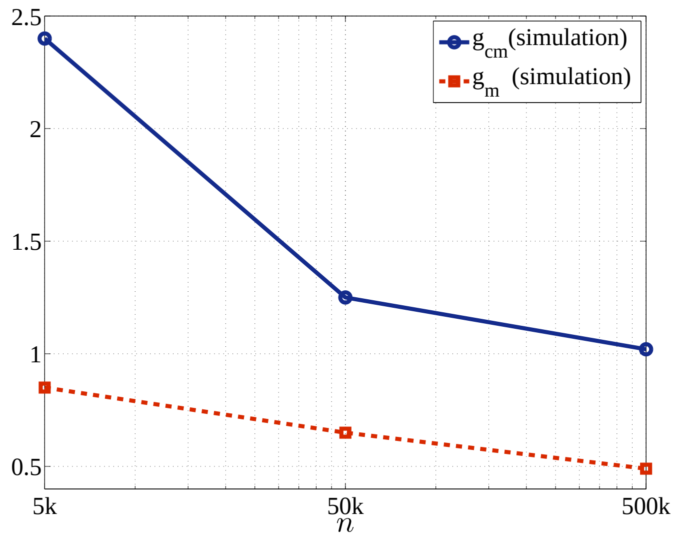


Figure 11: Simulation results for the compression gains g_{m} and g_{cm}.

the trend shown here for the compressible sequences would remain intact for different traces. Furthermore, if we also consider the incompressible part of the data, the resulting compression rate would be slightly worse, but still considerably better than Ucomp and UcompM.

3.5.4 Discussion on Complexity

3.5.4.1 Compression Complexity in Theory

The memory-assisted compression has a complexity linear in the packet size. The compressibility determination based on entropy estimation is also a linear operation in size of packets as it only entails the computation of empirical entropy of each packet. Thus far, these operations can be optimized to be performed on the packets on the fly. On the other hand, the K-means clustering is a more complex operation that needs to be performed offline. After the cluster centers are determined, which can be done every once in a while, the classification can also be done efficiently in linear complexity with the packet size.

The error-handling operation in the decompression setting deals with the cases

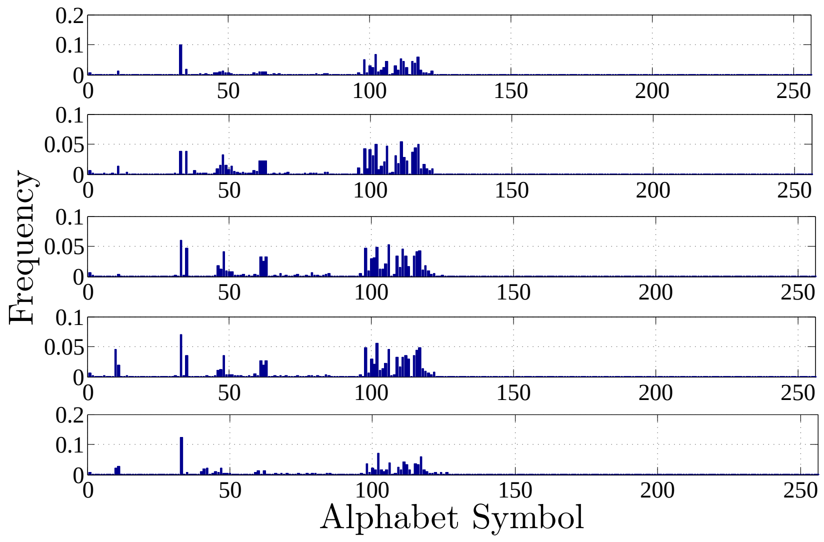


Figure 12: Empirical symbol distribution of cluster centers.

where the ID of the cluster (side information) a packet is compressed with does not exists in the memory of the decompressor. In such cases, the error handling module would send a feedback to the sender to request the cluster center to add it to the set of models.

3.5.4.2 Compression Complexity in Practice

The speed and performance of different compression algorithms varies widely. The statistical compression algorithms are tailored to offer superior compression performance, however, the compression speed of this class of compression algorithms is considerably lower than dictionary-based compression algorithms. There are several high-speed dictionary based compression algorithms, such as [51, 39, 40, 62]. These algorithms are the key part of some of the massive parallel computation systems, for example, Snappy [39] is used in Google infrastructure. The main goal in the design of such high-speed algorithms has been to adapt LZ77 compression to achieve highest

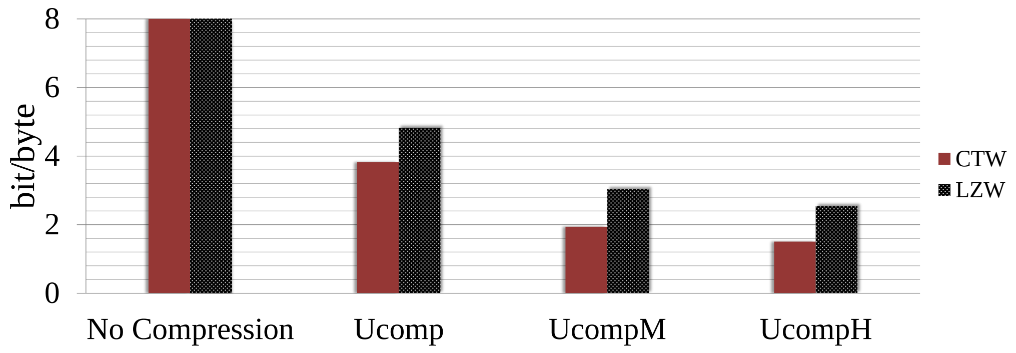


Figure 13: Results of various memorization and clustering schemes on compressible packets after entropy classification.

possible speed and through this process compression performance is traded for speed. As such, the compression performance of high-speed algorithms suffers, for example, compression of the first 1GB of the English Wikipedia using Snappy [39], and Gipfeli [51] has resulted in 530MB (in 2.8sec) and 410MB (in 4.3sec), respectively. However, a typical dictionary-based (Zlib) would compress the same input to 320MB (in 41.7sec). [3] In contrast, PAQ8, CTW, and lite PAQ compress the 1GB English text input to 134MB (in ≈30 ksec), 211MB (in ≈13 ks), and 164MB (in ≈1 ksec), respectively.

The high-speed compression algorithms are suitable for network compression in wired networks where the communication throughput is high, e.g., 128MB/sec for Ethernet 1 Gigabit/sec connection). The high-performance statistical compression algorithms are more suitable for wireless networks where the communication speed is lower (6.5 MB/sec for 802.11g) and higher compression rates are desirable.

[3]The execution time for dictionary-based algorithms is measured on an Intel Xeon W3690 CPU. The execution time of the statistical compression methods is measured on a Intel core i5 processor using only one of the cores.

3.6 Non-parametric Clustering and Infinite Mixture Models

The Hellinger distance metric proposed in Section 3.5 for clustering cannot be extended to complex statistical models. Furthermore, the k-means clustering algorithm requires the number of clusters to be known apriori which is not the case in general. To address this limitation, in Section 3.6, a principled clustering algorithm is presented that does not need to know the number of clusters in advance. Moreover, the use of Hellinger distance is relaxed and a more general feature extraction scheme is presented which is tailored to state-of-the-art statistical compression algorithms.

3.6.1 Infinite Mixture Models

In standard clustering algorithms like K-means clustering, it is assumed that the number of clusters is fixed and known. However, in many real-world examples the number of clusters is unknown. To address this problem, non-parametric Bayesian models [61, 35, 19, 36] are proposed that can potentially have infinite number of hidden clusters which naturally arise as more data become available. In other words, instead of fixing the number of clusters to be discovered, the number of clusters is allowed to grow as more data comes in.

The statistical process used for modeling the data generation and then clustering is Dirichlet Process (DP) mixture model. DP mixture models are the cornerstone of non-parametric Bayesian statistics [19, 82]. Figure 14 depicts the graphical model representation of a DP mixture model used in the section. Nodes denote random variables, edges denote possible dependence, and plates denote replication. The mixture model considered for generating data is assumed to have an infinite number of sources where k-th source is described by a set of parameters θ_k. The source parameters are assumed to come from a base distribution denoted by G_0. In the DP mixture, the vector π is a vector defined as

$$\pi_k(\mathbf{v}) = v_k \prod_{j=1}^{k-1}(1 - v_j),$$

where each v_j is a draw from Beta distribution with parameter α, i.e., $v_j \sim \text{Beta}(1, \alpha)$. Let z_i be the index of the mixture component with which the data point x_i is associated. The data generating process can be described as below [19]:

- $\theta_k \sim G_0$
- for i-th data point:

  (a) Draw $Z_i \sim \mathbf{V} = \text{Mult}(\pi(\mathbf{v}))$.

  (b) Draw $X_i \sim \mathbf{P}[x_i|\theta_{z_i}]$.

Data generated from this process can be partitioned according to the distinct values of the sources. It is observed from this model that the number of mixture components is random.

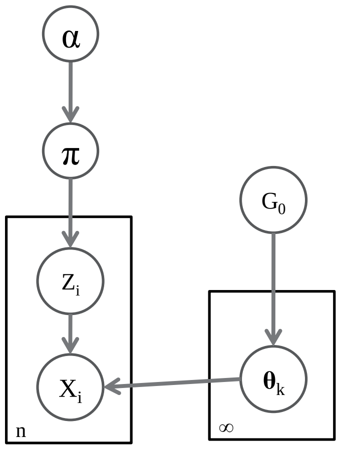


Figure 14: Graphical model representation of Dirichlet Process mixture model.

3.6.2 Inference

Expectation Maximization [18] is generally used for inference in a mixture model. However, with the non-parametric description of the model, Expectation Maximization is not directly applicable. As such, two different classes of inference techniques are used, namely, class of Markov Chain Monte Carlo (MCMC) techniques [57], and class of variational inference techniques [19]. Both MCMC and variational inference methods have their roots in statistical physics. MCMC are a class of algorithms for sampling from probability distributions based on constructing a Markov chain that has the desired distribution as its equilibrium distribution. The state of the chain after a large number of steps is then used as a sample of the desired distribution. The quality of the sample improves as a function of the number of steps. MCMC methods can be slow to converge and their convergence can be dicult to diagnose. As such, in some of the practical implementations [59] variational methods are used for inference. The basic idea of variational inference is to formulate the computation of a marginal or conditional probability in terms of an optimization problem. This problem is often intractable and to arrive at a solution, the problem is then relaxed, yielding a simplified optimization problem that depends on a number of free parameters, known as variational parameters. Solving for the variational parameters gives an approximation to the marginal or conditional probabilities of interest. In short, variational methods are a class of deterministic algorithms that convert inference problems into optimization problems [83].

The optimization problem in variational inference is constructed as follows. Consider a model with latent variables $\mathbf{w} = \{\mathbf{Z}, \mathbf{V}, \theta\}$ that are controlled with a known hyperparameter α. The observations are $x^n = \{x_1, \dots, x_n\}$. The posterior distribution of the latent variables is

$$\log \mathbf{P}[\mathbf{w}|x^n] = \log \mathbf{P}[\mathbf{w}, x^n] - \log \mathbf{P}[x^n]. \tag{12}$$

The variational method is based on optimizing KL divergence with respect to a so-called variational distribution, denoted by $\mathbf{Q}[\mathbf{w}]$. The optimization problem aims to minimize the KL divergence between $\mathbf{Q}[\mathbf{w}]$ and $\mathbf{P}[\mathbf{w}|x^n]$. We have

$$\begin{aligned} D\big(\mathbf{Q}[\mathbf{w}] \parallel \mathbf{P}[\mathbf{w}|x^n]\big) &= \mathbf{E}_Q\big[\log \mathbf{Q}[\mathbf{w}]\big] - \mathbf{E}_Q\big[\log \mathbf{P}[\mathbf{w}, x^n]\big] + \log \mathbf{P}[x^n] \quad (13) \\ &= \mathbf{E}_Q\big[\log \mathbf{Q}[\mathbf{w}]\big] - \mathbf{E}_Q\big[\log \mathbf{P}[\theta]\big] - \mathbf{E}_Q\big[\log \mathbf{P}[\mathbf{V}]\big] \\ &\quad - \sum_{i=1}^{n} \big\{\mathbf{E}_Q\big[\log \mathbf{P}[Z_i|\mathbf{V}]\big] + \mathbf{E}_Q\big[\log \mathbf{P}[x_i|Z_i]\big]\big\} \\ &\quad + \log \mathbf{P}[x^n]. \quad (14) \end{aligned}$$

In the variational framework the distribution $\mathbf{Q}(.)$ is chosen such that the optimization problem in (13) is tractable. In practical implementations, a fully-factorized variational distributions is often considered which break all of the dependencies between latent variables and the final problem becomes tractable.

In [19], it is shown that by choosing a fully-factorized $\mathbf{Q}(.)$ one can arrive at an optimization problem which can be solved using standard gradient descent techniques. The efficiency of this approach is demonstrated in [19] and an implementation of this technique can be found in [59]. The complexity of the implementation in [59] is linear in the number of mixture components and data points. The approximate inference algorithm in the implementation uses a truncated distribution with a fixed maximum number of mixture components, but almost always the number of components actually used depends on the data.

3.7 Extracting Features From Statistical Compression Methods for Clustering

The non-parametric methods discussed in Section 3.6 provide a statistically structured way of clustering the data points even when the actual number of clusters is unknown. In Section 3.7, a new approach of feature extraction from statistical compressors is presented. This approach lends itself to the the non-parametric clustering and as a result superior compression performance is demonstrated.

As previously discussed in Section 2.3, the statistical compressor consists of two parts: a predictor and an entropy coder, as shown in Figure 42(a). Let $x^n = (x_1, \ldots, x_n)$, $x_t \in \mathcal{A}$ be the input packet. The compressor processes x^n sequentially such that at t-th step the predictor emits the probability distribution of the next symbol, denoted by $\mathbf{P}[x_t|x_1^{t-1}]$, based on the already processed sequence $x_1^{t-1} = x_1, \ldots, x_{t-1}$. Given the probability distribution, the entropy coder maps x_t to a codeword of a length close to $-\log \mathbf{P}[x_t|x_1^{t-1}]$ bits. The predictor, as depicted in Figure 15, is composed of a number of simple models. Each model in predictor issues an estimate of the probability distribution of the next symbol. Let $e_1, \ldots, e_p$ be the estimate of models $M_1, \ldots, M_p$, respectively.

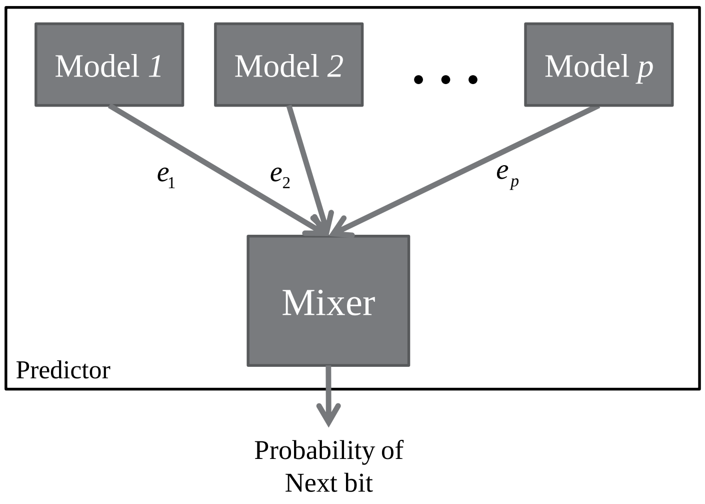


Figure 15: The internal design of the predictor.

It is expected that simple models in the predictor provide consistently similar estimates for data sequences that have similar statistical properties. As a result, if we only use a single model in isolation, the expected estimated code length of that model for sequences with similar structure would be concentrated around each other. This provides the basic of the feature extraction. As such, the approach presented for feature extraction is as follows: Given a set of input packets, each packet is fed into the models $M_1, \ldots, M_p$ separately. The sequence of the estimates of each model is then transformed to an estimated code length for the input packet. For example, consider the sequence of the estimates generated by the j-th model M_j, i.e., $e_j(t)$, $t = 1, \ldots, n$. The estimated code length of M_j, denoted by l_j, is

$$l_j = \sum_{t=1}^{n} - \log e_1(t), \qquad \text{for } j = 1, \ldots, p.$$

At this point, with every packet a feature vector $(l_1, \ldots, l_p)$ is associated. These feature vectors provide the data point for the clustering algorithm. More details on the internal operations of the PAQ statistical compression method is presented in Appendix A.

3.7.1 Experiment Results

For experimentation, similar to setup in Section 3.5.3, real-world packets captured from CNN and Apple websites are used for experiments. The IP packets of size ≈1.5 kByte are captured using the "wireshark" packet analyser. The total size of the data traces captured from both websites is 160MB. The CNN and Apple data is referred to as "data set 1" and "data set 2", respectively.

The first 90% of the data packets in the beginning of each data trace are chosen as training set and the remaining 10% are test packets. The structure of the data packets (whole data including training and test packets) in each trace is depicted in Figure 16. Each pixel represents a duplicate string (consecutive byte sequence). The color of the pixel represents the length of the match: black for 1 byte, red for 2, green for 4 and blue for 8. The visualizations are generated using the "fv" tool [1] developed by Matt Mahoney. These visualizations clearly demonstrate the varying patterns observed in the data sequence.

The results of the experiment are presented in Table 1. The first two rows are the result of compression without using memory for both statistical and dictionary-based algorithms. The dictionary-based compression scheme used is Gzip on Linux using the default options and the statistical compression is lite PAQ (Appendix A) with option 4. [4] For clustering, the implementation of non-parametric clustering from Scikit Learn library [59] is used. Once again, it is observed that statistical compression outperforms the dictionary-based compression. Furthermore, the effectiveness of the clustering algorithm combined with the statistical compression is demonstrated and it is observed that, on the average, memory-assisted compression and clustering provides a factor of 2 improvement over the Ucomp. Moreover, the non-parametric clustering with the features vectors chosen as the code length of the simple models provides the same level of performance improvement as Hellinger-distance clustering (Section 3.5) with considerably less complexity.

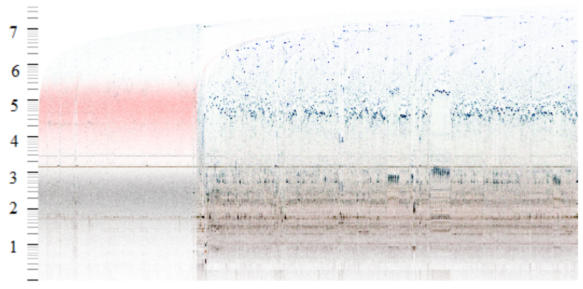
Table 1: The compression rate of data set 1 and data set 2 with different coding strategies. The statistical compression scheme used is lite PAQ.

Compression Scheme	**compression rate** (bits/byte)	
	Data set 1	Data set 2
Ucomp (Dic.)	6.13	5.35
Ucomp (Stat.)	5.62	4.83
UcompM (no clustering)	2.77	2.71
UcompM with non-parametric clustering	2.64	2.47

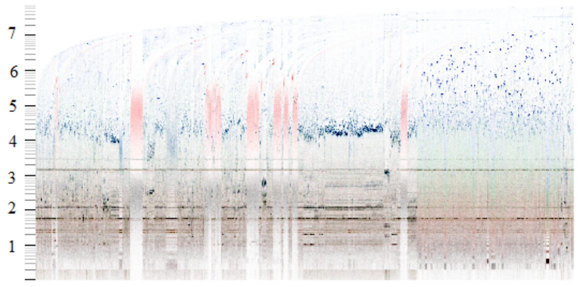
3.7.2 A Case Study on Detaching Training From Compression

In Section 3.7.2, we investigate whether it is possible to detach the clustering from compression. In other words, it is of interest to find out whether training and compression can be performed on different and independent data sets. The objective of this separation, if possible, is to train a joint statistical and clustering module on a large data set and use the result of that training for compression of a wide range of test data without the need to repeat the training. The results of our experiment are

[4]The option determines the amount of storage used by the algorithm. Option 4 used 50MB of storage.



(a) Data set 1 (CNN)



(b) Data set 2 (Apple)

Figure 16: Visualization of the data traces. Each pixel represents a duplicate string (consecutive byte sequence). The color of the pixel represents the length of the match: black for 1 byte, red for 2, green for 4 and blue for 8. The horizontal axis represents the position of the second occurrence of the string from the beginning. The vertical axis represents the distance back to the match on a $\log_{10}$ scale

presented in Table 2. The setup of the experiment is as follows. We consider the data sets introduced in Section 3.7.1. The data sets are divided into two parts each; the first 90% of the packets are considered as training data and the rest is test data. We have considered training with data set 1 and testing with data set 2 and vice versa. Both naive memory-assisted compression and memory-assisted compression with clustering is considered. The number of clusters that is assigned by non-parametric clustering algorithm to each data set is 5 and 6 for data set 1 and data set 2, respectively. As shown in Table 2, the clustering provides a modest improvement over naive memory-assisted compression performance. However, comparing the compression rate with that of Table 1, it is observed that, on the average, the detaching has about 80% toll on the compression performance. This performance degradation discourages detaching the training and compression in applications where compression performance is of importance.

Table 2: The compression performance of memory-assisted compression when training and test packets are chosen from different data sets.

Train	**Test**	**Scheme**	**Compression rate (bits/Byte)**
Data set 2	Data set 1	UcompM	5.07
Data set 2	Data set 1	UcompM+clustering	4.89
Data set 1	Data set 2	UcompM	4.43
Data set 1	Data set 2	UcompM+clustering	4.34

We can name the sensitivity of the statistical compression algorithm to the training as the main factor that contributes to the inefficiency of detaching the training and compression. In other words, if the statistical properties of the training and test data differ, this difference would result in diminishing compression performance and undermines the benefits of memory-assisted compression. To verify this hypothesis, we have considered two text documents written in English. These documents are BOOK1 (a book in ASCII text titled “Far from the Madding Crowd”) and BOOK2 (ASCII text titled “Principles of Computer Speech”) from Calgary Corpus [2]. The 90% in the beginning of each file is considered as training and the remaining 10% is considered as test data. The test data is split into 10kByte parts and the results are averaged over the parts. In Table 3, the experiment results are summarized.

Table 3: The compression performance of memory-assisted compression on two English text files.

Train	**Test**	**Compression rate (bits/Byte)**
BOOK1	BOOK1	1.94
BOOK1	BOOK2	2.04
BOOK2	BOOK1	2.30
BOOK2	BOOK2	1.70

The results presented in Table 3 indicate that detaching training from compression would increase the compression rate by 18% for BOOK1 and by 20% for BOOK2.

Compressing the test part of the data without memory (Ucomp) results in compression rate of 3.09 bits/Byte for BOOK1 and 2.82 bits/Byte for BOOK2. Therefore, the gain of memory-asisted compression for BOOK1 and BOOK2 is 1.6 and 1.65, respectively. However, after detaching training from compression would reduce the gain of memory-assisted compression reduces to 1.5 and 1.2 for BOOK1 and BOOK2.

From the experiment above it can be concluded that even for data types that are similar to each other, e.g., human-readable English text in ASCII format, the sensitivity of the statistical compression to training data would have a negative impact on the performance of the memory-assisted compression when training is detached from compression.

CHAPTER IV

NETWORK COMPRESSION IN WIRED NETWORKS: ERDŐS-RÉNYI RANDOM NETWORK GRAPHS

4.1 Introduction

In Chapter 4 and Chapter 6, the benefits of deploying memory units that enable memory-assisted compression in wired networks are studied. The goal is to quantify these benefits in terms of the amount of traffic reduction when memory-assisted compression is employed in a network. The question to be answered is "Given the memory-assisted source coding gain g, and a number of nodes capable of performing compression, what is the achievable network-wide compression gain of memorization?"

4.1.1 Contributions

In Chapter 4, the gain of network compression in the Erdős-Rényi (ER) network graph family is studied [70]. Analogous to the memory-assisted compression gain for a single link, network-wide gain of compression (gain of network compression in short) is defined and studied in ER random network graphs. The gain of network compression depends on the number of memory units; it is discovered that there exists a threshold value for the number of memories deployed in a random graph below which the network-wide gain of memorization vanishes.

In what follows, the necessary notions and the setting of the problem is introduced and then the results on gain of network compression in ER graphs is presented.

4.2 Notation

A network is represented by an undirected graph $G(V, E)$ where V is the set of N nodes (vertices) and $E = \{uv : u, v \in V\}$ is the set of edges connecting nodes u and v. We consider a set of memory units $\boldsymbol{\mu} = \{\mu_i\}_{i=1}^{M}$ chosen out of N nodes where every memory node μ_i is capable of memorizing the communication passing through it. The total size of memorized sequences for each μ_i is assumed to be equal to m. In this work, we focus on the expected performance of the network by averaging the gain over all scenarios where the source is chosen to be any of the nodes in the network equally at random. In other words, we assume that the source is located in any node of the network with probability $\frac{1}{N}$. As discussed before, end-to-end compression techniques will only be able to compress the content to a value which may be significantly larger than the entropy of the data sequence.

After the memorization phase, we can assume the constructed source model is available to all memory units. In the second phase, which is the subject of this section, we assume each node in the entire network may request content from the source. The above view simplifies our study as we are not concerned with the transition phase during which the memorization is taking place in the memory units. Hence, we can assume that each memory unit will provide the same memory-assisted compression gain of g on the link from the origin node of the flow to itself. The goal of the network compression is to minimize the total cost of communication between the source and destinations in the network, measured by bit×hop. As will be discussed below, the total cost of communication will depend on the memory-assisted gain g.

Consider the outgoing traffic of one of the nodes in the network, named as S, with the set of its destinations $\boldsymbol{D} = \{D_i\}_{i=1}^{N-1}$ each receiving different instances of the source sequence originated at S. Let f_D be the unit flow from S destined to $D \in \boldsymbol{D}$. The distance between any two nodes u and v is shown by $d(u, v)$. The distance is measured as the number of hops in the lowest cost path between the two nodes. As we will see later, introducing memories to the network will change the lowest cost paths, as there is a gain associated with the $S-\mu$ portion of the path. Therefore, we have to modify paths accounting for the gain of memories. Accordingly, for each destination D, we define *effective walk*, denoted by $W_D = \{S, u_1, \dots, D\}$, which is the ordered set of nodes in the modified (lowest cost) walk between S and D. Finding the shortest walk is the goal of routing problem with memories.

We partition the set of destinations as $\boldsymbol{D} = \boldsymbol{D}_1 \cup \boldsymbol{D}_2$, where $\boldsymbol{D}_1 = \{D_i : \exists \mu_{D_i} \in W_{D_i}\}$ is the set of destinations observing a memory in their effective walk, and $\mu_{D_i} = \arg\min_{\mu \in \boldsymbol{\mu}} \{\frac{d(S,\mu)}{g} + d(\mu, {\scriptstyle D_i})\}$. The total flow $\mathcal{F}_S$ of node S is then defined as

$$\mathcal{F}_S \triangleq \sum_{D_i \in \boldsymbol{D_1}} \left(\frac{f_{D_i}}{g} d(S, \mu_{D_i}) + f_{D_i} d(\mu_{D_i}, {\scriptstyle D_i}) \right) + \sum_{D_j \in \boldsymbol{D_2}} f_{D_j} d(S, D_j). \tag{15}$$

Using (15), we define $\hat{d}_D$, called the *effective distance* from S to D, as

$$\hat{d}_D = \begin{cases} \frac{d(S,\mu_D)}{g} + d(\mu_D, {\scriptstyle D}) & D \in \boldsymbol{D}_1 \\ d(S, D) & D \in \boldsymbol{D}_2 \end{cases}. \tag{16}$$

In short, the effective distance is the distance when memory-assisted compression is performed and hence the gain g applies. By definition, $\hat{d}_D \leq d(S, D)\ \forall D$.

In a general network, the generalized network-wide gain of network compression as a function of memorization gain g is defined as follows:

$$\mathcal{G}(g) \triangleq \frac{\sum_{S \in V} \mathcal{F}_S^0}{\sum_{S \in V} \mathcal{F}_S} = \frac{\sum_{S \in V} \sum_{D \in \boldsymbol{D}} d(S, D)}{\sum_{S \in V} \sum_{D \in \boldsymbol{D}} \hat{d}_D}, \tag{17}$$

where $\mathcal{F}_S^0$ is the total flow in the network by node S without using memory units, i.e., $\mathcal{F}_S^0 = \sum_{D \in \boldsymbol{D}} d(S, D)$. In other words, $\mathcal{G}$ is the gain observed in network achieved by memory-assisted scheme on top of what could be saved by an end-to-end compression scheme at the source without using memory. Alternatively, $\mathcal{G}(g)$ can be rewritten as

$$\frac{\sum_{S\in V}\sum_{D\in \boldsymbol{D}} d(S,D)\mathbf{E}l_n(X^n)}{\sum_{S\in V}\sum_{D\in \boldsymbol{D}}\left[d(S,\mu_D)\mathbf{E}l_{n|m}(X^n)+d(\mu_D, {\scriptstyle D})\mathbf{E}l_n(X^n)\right]}.$$

To show the challenges of the memory deployment problem, we show as to how a single memory changes the effective paths in a network with a single source. Consider the network with the source node S placed as shown in Figure 17. The destinations are nodes $D_1, \dots, D_4$, and $g = 4$. The effective walks from the source to destinations are obviously the shortest paths when there is no memorization (Ucomp). As shown in the figure, the placement of memory changes the effective path to D_2 while the shortest paths from the source to D_1, D_3, and D_4 are the same as the effective paths. Without memory, the shortest path to D_2 is two hops long ($S \to D_1 \to D_2$), while the memory totally changes the effective walk distance to D_2 to $\hat{d}_{D_2} = \frac{3}{4} + 1$ as depicted in the figure.

Next, we consider the gain of network compression in the network graphs that resemble the ER random graph [33]. The ER random graph is the building block of the recent models for complex graphs and hence the results would be useful in much broader contexts. We specifically direct our attention to connected random graphs since they better describe real networks.

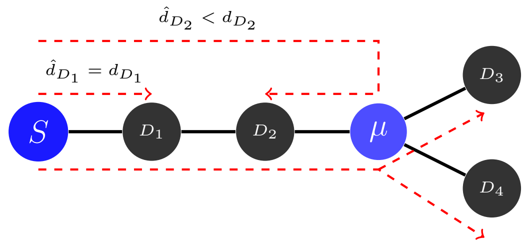


Figure 17: The shortest walk between the source and destination is not necessarily a path when there are memory units in the network.

4.3 Gain of Network Compression in ER Random Graphs

4.3.1 Background on ER Random Graphs

Definition An ER random graph $G(N,p)$ is an undirected, unweighted graph on N vertices where every two vertices are connected with probability p.

Definition Let $u, v \in G$ be any two vertices. The diameter of a connected graph is defined as $\max_{u,v} d(u,v)$. Similarly, the average distance of a connected graph is defined as $\mathbf{E}[d(u,v)]$.

The following properties hold for ER random graphs:

1. $G(N,p)$ has on average $\binom{N}{2}p$ edges.

2. If $p < \frac{(1-\epsilon)\log N}{N}$, then $G(N,p)$ almost surely (a.s.) has isolated vertices and thus disconnected.

3. If $p = \frac{c \log N}{N}$ for some constant $c > 1$, then $G(N,p)$ is a.s. connected and every vertex asymptotically has degree $c \log N$ [5].

4. The diameter of $G(N,p)$ is almost surely $\frac{\log N}{\log Np}$.

5. The average distance in $G(N,p)$, denoted by $\bar{d}$, is

$$\bar{d} = (1 + o(1))\frac{\log N}{\log Np}, \tag{18}$$

   provided that $\frac{\log N}{\log Np}$ goes to infinity as $N \to \infty$ (this condition is satisfied in the connected regime) [28].

4.3.2 Main Result

The main question is how $\mathcal{G}(g)$ scales with the number of memories M. To characterize the gain of network compression, we consider connected $G(N,p)$, $p = \frac{c \log N}{N}$, with a single source node S and all other nodes as destinations. Since the expected degree of all nodes in ER graph is the same and every vertex is a destination with equal probability, we select memories $\{\mu_i\}_{i=1}^{M}$ uniformly at random. Theorem 4.3.1 provides the scaling of $\mathcal{G}(g)$ with respect to M.

Theorem 4.3.1 *Suppose M is the number of deployed memories in an ER random graph. Let ϵ be a positive real number.*

(a) If $M = O\left(N^{\frac{1}{g}-\epsilon}\right)$, then $\mathcal{G}(g) \sim 1$. [1]

(b) If $M = \Omega\left(N^{\frac{1}{g}+\epsilon}\right)$, then $\mathcal{G}(g) \sim \frac{g}{1-g\log_N(\frac{M}{N})}$.

Sketch of the proof We first find an upper bound on the number of destinations benefit form each memory. This upper bound is sufficient to derive part (a) of the theorem. For the second part, we find a lower bound on the number of benefiting destinations. ■

To characterize $\mathcal{G}(g)$, we first need to find $\mathcal{F}_0$. The average distance from the source to a node is $\bar{d}$. Thus, $\mathcal{F}_0 = N\bar{d}$. For large N, (18) results in

$$\mathcal{F}_0 \sim \frac{N \log N}{\log \log N}.$$

Next, we need to find $\mathcal{F}$. For every memory μ we consider a neighbourhood $\boldsymbol{N}_r(\mu)$ as shown in Figure 18. This neighborhood consist of all vertices v within distance r from μ. We choose r such that, almost surely, all nodes in $\boldsymbol{N}_r(\mu)$ would benefit from the memory μ. Clearly, if $\frac{d(S,\mu)}{g} + r = d(S, v)$, the benefit of the memory for node v

[1]Throughout this work, we have used the following asymptotic notations:

– $f(x) = o(g(x))$ iff $|f(x)| \leq |g(x)|\epsilon$, $\forall\epsilon$,

– $f(x) = O(g(x))$ iff $|f(x)| \leq |g(x)|k$, $\exists k$,

– $f(x) = \Omega(g(x))$ iff $|f(x)| \geq |g(x)|k$, $\exists k$, and

– $f(x) \sim g(x)$ iff $f(x)/g(x) \to 1$.

vanishes and only nodes at distances less than r benefit from the memory μ. Given g, we denote this set of nodes benefiting from μ by $\boldsymbol{N}_r(\mu, g)$.

$$\boldsymbol{N}_r(\mu, g) = \left\{ v : \frac{d(S,\mu)}{g} + d(\mu, v) \le d(S, v) \right\}. \tag{19}$$

Since memories are uniformly placed, the average value of $d(S,\mu)$ in $\hat{d}_v$ is equal to $\bar{d}$. Similarly, the average of $d(S,v)$ is also $\bar{d}$. Hence, solving for r in (19) and then using the result on the average distance in (18), we conclude

$$r \overset{a.s.}{=} (1 - 1/g)\left(\frac{\log N}{\log\log N}\right). \tag{20}$$

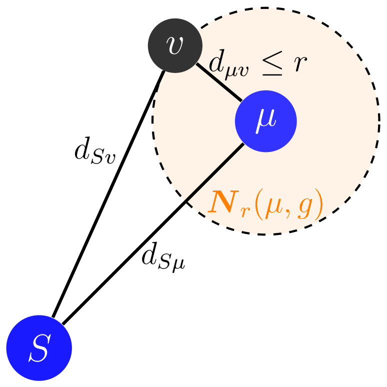


Figure 18: Illustration of Memory Neighborhood.

The following lemma, by Chung and Lu [28], gives an upper bound on the total number of vertices in the neighborhood $|\boldsymbol{N}_r(\mu_i, g)|$, where $|\cdot|$ is the set size operator.

Lemma 4.3.2 ([28]) *Assume a connected random graph. Then, for any $\epsilon > 0$, with probability at least $1 - \frac{1}{(\log N)^2}$, we have $|\boldsymbol{N}_r(\mu_i, g)| \le (1+2\epsilon)(Np)^r$, for $1 \le r \le \log N$.*

Using Lemma 4.3.2 and (20), we deduce that

$$\begin{aligned} |\boldsymbol{N}_r(\mu_i, g)| &\overset{a.s.}{\le} (1+2\epsilon)(\log N)^{(1-\frac{1}{g})\left(\frac{\log N}{\log\log N}\right)} \\ &= (1+2\epsilon)N^{1-1/g}. \end{aligned} \tag{21}$$

Therefore, the total number of nodes gaining from the memories is upper-bounded by $\sum_{i=1}^{M} |\boldsymbol{N}_r(\mu_i, g)| \leq M(1+2\epsilon)N^{1-1/g}$. As we will see, from (21) it is clear that the gain of memory vanishes if M is chosen small. The value $N^{1/g}$ is the threshold value for the network-wide gain. More accurately, if $M = O\left(N^{\frac{1}{g}-\epsilon}\right)$, there is no gain from memories.

4.3.3 Proof of the Main Result

Proof of Theorem 4.3.1(a) For all the nodes in $\boldsymbol{N}_r(\mu_i, g)$, we have a flow gain of g. Let $M = N^{\frac{1}{g}-\epsilon}$, then we have

$$
\begin{aligned}
\mathcal{G}(g) &\leq \frac{N\bar{d}}{\frac{\bar{d}}{g}M|\boldsymbol{N}_r(\mu, g)| + \bar{d}(N - M|\boldsymbol{N}_r(\mu, g)|)} && (22)\\
&\overset{a.s.}{\leq} \frac{N}{N - (1-1/g)MN^{(1-\frac{1}{g})}} && (23)\\
&= \frac{N}{N-(1-1/g)N^{1-\epsilon}} \sim 1,
\end{aligned}
$$

where inequality in (22) follows from the double counting of the destination nodes that may reside in more than one neighborhood. Also, (23) follows from replacing (21) in (22). ■

Since we need more than $n^{\frac{1}{g}}$ memory units to have a network-wide gain, the next question is as to how $\mathcal{G}(g)$ scales when the number of memory units exceeds $n^{\frac{1}{g}}$. To answer this question, we need to establish a lower-bound on the neighborhood size and the number of nodes benefiting from memory. Further, we have to account for the possible double counting of the intersection between the memory neighborhoods. We use the following concentration inequality from [28] to establish the desired bound.

Proposition 4.3.3 ([28]) *If $X_1, X_2, \ldots, X_n$ are non-negative independent random* *variables, then the sum* $X = \sum_{i=1}^{n} X_i$ *holds the bound*

$$\boldsymbol{P}[X \leq \mathbf{E}[X] - \lambda] \leq \exp\left(-\frac{\lambda^2}{2\sum \mathbf{E}[X_i^2]}\right).$$

This inequality will be helpful to show that the quantities of interest concentrate around their expected values.

The following lemma provides a lower-bound on the neighborhood size $|\boldsymbol{N}_r(\mu, g)|$ and the lower-bound on $\mathcal{G}(g)$, as we show, is immediate.

Lemma 4.3.4 *Consider a set of vertices* V *of* $G(N, p)$ *such that* $\frac{|V|}{N} = o(1)$. *For* $0 < \epsilon < 1$, *with probability at least* $1 - e^{-Np|V|\epsilon^2/2}$, *we have*

$$|\boldsymbol{N}_r(\mu, g)| \geq (1 - \epsilon)(Np)^r. \tag{24}$$

Proof The vertex boundary of V, denoted by $\Gamma(V)$, consists of all vertices in G adjacent to some vertex in V.

$$\Gamma(V) = \{u : u \notin V, \text{ and } u \text{ is adjacent to } v \in V\}.$$

Let X_u be the indicator random variable that a vertex u is in $\Gamma(V)$, i.e., $\boldsymbol{P}[X_u = 1] = \boldsymbol{P}[u \in \Gamma(V)]$. Then,

$$\begin{aligned}
\mathbf{E}\left[|\Gamma(V)|\right] &= \sum_{u \notin V} \mathbf{E}[X_u] = \sum_{u \notin V} \boldsymbol{P}[u \in \Gamma(V)] \\
&= \sum_{u \notin V} \left(1 - (1-p)^{|V|}\right) \\
&\geq p|V|(N - |V|) = (1 - o(1))Np|V| \qquad (25)
\end{aligned}$$

where the inequality in (25) follows from

$$\boldsymbol{P}[u \in \Gamma(V)] = 1 - (1-p)^{|V|} \geq 1 - e^{-p|V|} \approx p|V|,$$

and the second part holds because $\frac{|V|}{N} = o(1)$. Since, X_u's are non-negative independent random variables, by applying Proposition 4.3.3 with $\lambda = \sqrt{\alpha \mathbf{E}[|\Gamma(V)|]}$, with probability at least $1 - e^{-\alpha/2}$ we have

$$\begin{aligned} |\Gamma(V)| &\geq \mathbf{E}\left[|\Gamma(V)|\right] - \sqrt{\alpha \mathbf{E}[|\Gamma(V)|]} \\ &\geq (1-\epsilon) N p |V|. \end{aligned} \tag{26}$$

By picking a single vertex and applying (26) inductively r times, and then adding up the number of adjacent nodes, we obtain (24). ■

Now that we have a lower-bound on the number of nodes benefiting from each memory, we show that by increasing the number of memories beyond $M = N^{\frac{1}{g}}$, memories cover all the nodes in the graph effectively and hence all the nodes would gain from the memory placement.

In order to limit the intersection between the neighborhoods, we reduce r to r_δ as below:

$$r_\delta = (1 - 1/g - \delta)\left(\frac{\log N}{\log \log N}\right). \tag{27}$$

With this choice of r_δ, by lemmas 4.3.2 and 4.3.4, we deduce that the probability that a random node $u \in G$ belongs to the neighborhood $\boldsymbol{N}_{r_\delta}(\mu_i, g)$ of the memory μ_i is $N^{-1/g-\delta}$. Hence, the expected number of the covered nodes is

$$\begin{aligned} \mathbf{E}\left[\left|\bigcup_{i=1}^{M} \boldsymbol{N}_{r_\delta}(\mu_i, g)\right|\right] &= \sum_{u \in G} \boldsymbol{P}\left[u \in \cup_{i=1}^{M} \boldsymbol{N}_{r_\delta}(\mu_i, g)\right] \\ &= \sum_{u \in G} \left(1 - (1 - N^{-1/g-\delta})^{M}\right) \\ &\approx N\left(M N^{-1/g-\delta}\right) = N, \end{aligned} \tag{28}$$

where (28) holds by choosing $M = N^{1/g+\delta}$.

To show that the number of covered nodes is concentrated around its mean, we use Proposition 4.3.3 again with $\lambda = \sqrt{\alpha \mathbf{E}\left[|\cup \boldsymbol{N}_{r_\delta}(\mu_i, g)|\right]}$. Then, with probability at least $1 - e^{-\alpha/2}$ we have

$$\begin{aligned} \left|\bigcup_{i=1}^{M} \boldsymbol{N}_{r_\delta}(\mu_i, g)\right| &\geq \mathbf{E}\left[|\cup \boldsymbol{N}_{r_\delta}(\mu_i, g)|\right] - \lambda \\ &\geq (1 - o(1))N. \end{aligned}$$

Hence, the memories cover, almost surely, all of the nodes.

Since all nodes are covered with high probability, we can associate each node with a neighborhood $|\boldsymbol{N}_{r_\delta}(\mu_i, g)|$, for which nodes' distances in the neighborhood from memory are $(1 - o(1))r_\delta$.

Proof of Theorem 4.3.1(b) By (28), we can bound the network-wide gain of the memory from below. We have

$$\mathcal{G}(g) \overset{a.s.}{=} \frac{N\bar{d}}{(\bar{d}/g + r_\delta)N} \tag{29}$$

$$= \frac{1}{1/g + (1 - 1/g - \delta)} = \frac{1}{1 - \delta}, \tag{30}$$

where (29) holds because the distance of the nodes from memory is r_δ, asymptotically almost surely. ■

As the number of memories becomes close to N, i.e., $\delta \to (1 - \frac{1}{g})$, the gain $\mathcal{G} \to g$, as expected. In the next section, we verify our result in memory-assisted source coding and the network-wide gain of memory via numerical simulations.

CHAPTER V

ROUTING AND PLACEMENT PROBLEM IN NETWORKS WITH MEMORY

5.1 Introduction

The deployment of memory units in the network gives rise to a number of questions and also brings some new challenges. In network compression, every traffic from source to the memory node benefits from the gain g. In Chapter 4, we defined a network-wide gain $\mathcal{G}(g)$ for a general network topology as a function of the number of memory units in the network, and the fundamental gain g of memory-assisted source coding. In this chapter, we aim at answering two fundamental (and related) questions regarding memory-assisted network compression.

1. In practical scenarios where only a select number of nodes are capable of memorization and data processing, i.e., only certain number of nodes in the network are memory units, what would be the best strategy to choose the memory units? In other words, where should the memory units be placed in the network?

2. After the locations of the memory units are fixed, what would be the optimal routing algorithm? In other words, what is the best strategy to route packets between the source and destination nodes given the network topology, the location of the memories, and the fundamental gain g of memorization?

5.1.1 Contributions

The memory placement problem in non-random network graphs is studied in Chapter 5. It is shown that optimal memory-placement is not tractable in general network graphs and the challenges involved are demonstrated by deriving the optimal memory placement on line networks [74]. In Chapter 5, the routing problem for compressed flows is solved and a modified Dijkstra's algorithm is presented.

The memory placement in the network poses some challenges to traditional shortest path routing algorithms, as the shortest path is not necessarily minimum cost route in networks with memory. The well-known routing algorithms like Dijkstras algorithm, in their original form are not applicable to networks with memory. As such, in Chapter 6, the routing problem for compressed flows is considered and a modified Dijkstra's algorithm for compressed flows is presented.

5.2 Memory Placement Problem

The gain of network compression depends on the number of memory units and their locations. Since in practical scenarios only a select number of nodes have the storage and computational capability to function as a memory unit, it is important to find the optimal location for such nodes. In the ER random networks, we used a probabilistic treatment of the memory placement problem that was fit to the random nature of the ER graph. In this section, the memory placement problem is studied in a non-random graph and the challenges involved are presented. In particular, the optimal placement strategy on line networks is obtained.

Let the total number of memory units be M. The goal of the memory deployment is to find the best set of M out of N vertices in the network such that $\mathcal{G}(g)$, i.e., the network-wide gain of memory, is maximized. In general, this is a hard problem as we summarize below.

5.2.1 Hardness of Memory Placement Problem

It can be shown that the memory placement is equivalent to the well-known k-median problem. Hence, the memory deployment problem on a general graph is an NP-hard problem. However, a solution to the deployment problem can be obtained for certain network topologies, which can be helpful in finding approximate solutions for general networks. In this section, we demonstrate the challenges of the memory deployment problem by considering the class of line networks.

Another challenge in finding the gain $\mathcal{G}(g)$ comes from the difficulty of finding minimum cost paths in a network. Below we summarize these challenges and introduce a modified routing algorithm that can help finding the effective walks in a network and hence calculating $\mathcal{G}(g)$.

5.2.2 Memory Deployment on Line Networks

Consider a line network with the source node S placed at one end of the line and the destinations placed along the line as shown in Figure 19. Therefore, we have a total number of N nodes on the line and the total length of the line is N hops. As mentioned before, we assume traditional universal compression would give one unit of flow, to be sent to each destination. We consider the deployment of M memory units on the line such that the memory μ_i is placed at t_i from the source, as shown in Figure 19. We find t_i's such that total flow $\mathcal{F}$ is minimized (or equivalently, $\mathcal{G}(g)$ is maximized). The solution to the special case of "en-route" memory deployment on line networks is studied in [49]. En-route memories are those which are only located along routes from source to receivers. An en-route memory intercepts any request that passes through it along the regular routing path. The solution to the en-route memory placement problem as discussed in [49] is $t_i = \frac{i}{M} \quad \forall \mu_i$.

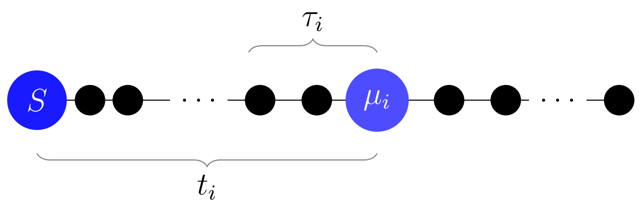


Figure 19: The placement of memory units on a line network: the source node S is placed at one end of the line and the i-th memory is placed at t_i from the source and τ_i is its left-coverage.

However, the memory deployment problem for network compression on a line network is more challenging. The difficulty comes from the fact that each memory can serve some of the destinations closer to source than the memory itself. In other words, the shortest effective walk from source to the destinations is not necessarily the same as the shortest hop distance. As shown in Figure 19, for a memory μ_i located at t_i, there is a left-coverage hop-length of τ_i towards the source to cover the destinations on the left side of the memory. The following lemma shows how t and τ change for different values of g.

Lemma 5.2.1 *For the simple case of $M = 1$ and a line of hop-length N, the optimal memory location t and coverage τ is given by*

$$\begin{aligned} t &= \frac{2g}{3g+1}N + O(1), \\ \tau &= \frac{g-1}{3g+1}N + O(1). \end{aligned} \tag{31}$$

Proof Using Fig 19, we can write the total flow as

$$\begin{aligned} \mathcal{F} = \int_0^{t-\tau} x\,\mathrm{d}x \quad &+ \quad \left(\frac{t(1-t)}{g} + \int_0^{1-t} x\,\mathrm{d}x\right) \\ &+ \quad \left(\frac{t\tau}{g} + \int_0^{\tau} x\,\mathrm{d}x\right) \end{aligned} \tag{32}$$

The first term in (32) is the flow to all points on the line not covered by memory. The second term is for the right coverage of memory and the third term accounts for the left coverage of memory (τ). The result in (31) follows by taking the derivative of $\mathcal{F}$ and equating to zero, i.e., $\frac{\partial}{\partial t}\mathcal{F} = 0$ and $\frac{\partial}{\partial \tau}\mathcal{F} = 0$. Figure 20 shows the plot of t and τ versus g. ∎

As shown in Figure 20, as the gain g increases, the memory is placed on 2/3 distance from the source and the left coverage approaches 1/3.

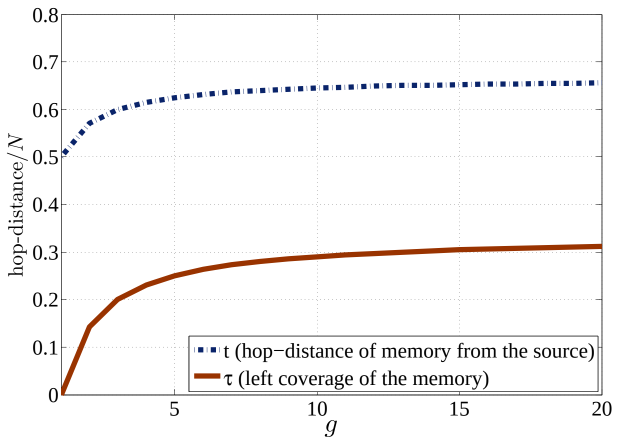


Figure 20: Variations of t and τ vs. g for a line network.

Lemma 5.2.2 *The gain of single memory placement on line is* $\mathcal{G}(g) = \frac{(3g+1)^2}{3g^2+10g+3}$, *and for* $g \gg 1$ *we have*

$$\mathcal{G} \approx 3. \tag{33}$$

Proof The proof is immediate from Lemma 5.2.1 and the fact that for line $\mathcal{F}_0 = 1/2$.

∎

Following the results of deployment of a single memory on line, we can extend the result and solve for the general problem of deployment of M memory nodes.

Theorem 5.2.3 *Consider deployment of* M *memories on a line where memory* μ_i *is placed at* t_i *with* τ_i *left-coverage. Then,*

$$\left\{ \begin{array}{lll} t_i & \approx & \frac{i}{M} N + O(1) \\ \tau_i & \approx & \frac{g-1}{2gM} N + O(1) \end{array} \right. . \tag{34}$$

By definition, $t_0 = \tau_0 = 0$. *Furthermore,* $\mathcal{G}(g) = \frac{2g^2 M}{2g(M+1)+g^2+1}$, *and for* $g \gg 1$ *we have*

$$\mathcal{G} \approx 2M. \tag{35}$$

Proof Similar to the proof of Lemma 5.2.1, we can write

$$\begin{aligned}\mathcal{F} &= \sum_{m=1}^{M} \Bigg[\frac{t_i}{g}\tau_i + \int_0^{\tau_i} x\,\mathrm{d}x \\ &\quad + \frac{t_i}{g}(t_{m+1} - \tau_{m+1} - t_i) + \int_0^{t_{m+1}-\tau_{m+1}-t_i} x\,\mathrm{d}x\Bigg].\end{aligned}$$

Again, by taking the derivative of $\mathcal{F}$ with respect to t_i and τ_i and solving the system of equations we arrive at

$$\begin{cases} t_i \approx \frac{t_{i+1}+t_{i-1}}{2} \\ \tau_i = \frac{g-1}{2g}\left(t_i - t_{i-1}\right) \end{cases}, \tag{36}$$

where (36) results in a tridiagonal matrix which in turn results in (34) for large M. Further, (35) follows from (34) and $\mathcal{F}_0 = 1/2$ for line networks. ∎

5.3 Routing in Networks Featuring Memory

Thus far, we have demonstrated that a non-vanishing network-wide gain is achievable by deploying a small number of memory units in the high degree nodes in RPLG. After the memory units are deployed, the next problem that arise is the routing in networks with memory units. In networks featuring memory, the source compresses the outgoing traffic and memory nodes decompress it, and hence there exists a compression gain between source and memory. This gain of compression poses new problems for conventional routing schemes. For example, as shown in Figure 21, let $g = 5$. Then, for node 2 the cost of routing clockwise is two, but it is only $\frac{9}{5}$ $(\frac{4}{5} + 1)$ by passing through memory unit μ counter-clock wise. Hence, introducing memory units in the network can dramatically change the effective shortest path. Routing is a new problem in networks featuring memory which needs to be solved differently. We consider an instance of network with a source node and fixed memory locations. We solve the routing problem in that instance of the network and after that we will be able to characterize $\mathcal{G}$.

Characterizing $\mathcal{G}$ involves computing both $\mathcal{F}^0$ and $\mathcal{F}$, which in turn requires finding the shortest paths between all pairs of nodes with and without memory units. The shortest path problem in a network without memories is straightforward via Dijkstra's algorithm. But finding shortest paths in networks featuring memory requires more attention. It is important to note that in network with memory if know the shortest path from a node u to v and also w is a node on the shortest path, this knowledge does not necessarily imply the knowledge of the shortest path from u to w.

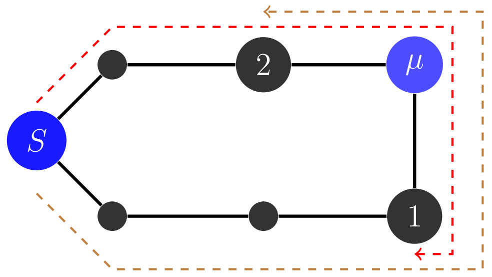


Figure 21: Example of routing in networks featuring memory: Memories can change the shortest paths (shown by dashed lines) dramatically. Here, $g = 5$.

The Dijkstra algorithm solves the single-source shortest path for a network with positive edge costs. Our bit×hop cost measure is a special case in which all the edge costs are equal to 1. However, in its original form, Dijkstra's algorithm is not applicable to networks with memory. In a regular network without memory nodes, the shortest path problem can be solved using the well-known Bellman-Ford algorithm which relies on the so-called principle of optimality: if a shortest path from u to v passes through a node w, then the portion of the path from w to v is also a shortest path. It is important to note that, there is another statement for principle of optimality as follows: if w is a node on the shortest path from u to v, this knowledge implies the knowledge of the shortest path from u to w. This latter statement only holds in networks without memory and in general is *not* true for networks with memory (Figure 21). Therefore, the shortest path problem requires more attention in networks with memory. The well-known routing algorithms like Dijkstra's algorithm, in their original form are not applicable to networks with memory. This limitation is due to the fact that when the algorithm runs into a memory node on the path, all the previous edges' costs along the path should be divided by g, and hence needs to recalculate the whole path.

Although, having memory units in a network changes the shortest paths dramatically (an example is shown in Figure 21), the principle of optimality sill holds and this will enable us to find the shortest path using the well-known Bellman-Ford algorithm which is in effect the repeated application of the principle of optimality. The Bellman-Ford algorithm is used in distance-vector routing protocols. The distributed version of the algorithm is used within an Autonomous System (AS), a collection of IP networks typically owned by an ISP. While Bellman-Ford algorithm solves the shortest path problem in networks with memory and the solution for routing within an AS is readily provided by this algorithm, the more efficient Dijkstra's algorithm is more used in practice. The Dijkstra's Algorithm is widely used in network routing protocols, most notably *IS-IS* and *OSPF* (Open Shortest Path First) and hence it is important to visit the challenges of finding the shortest paths in networks with memory using the Dijkstra's algorithm.

Here, we present a modified version of Dijkstra's algorithm that finds the effective walk from all the nodes in a network to a destination D, in a network with a single memory. Iterating over all nodes will provide the effective walk between every pair of nodes in the network. The extension to arbitrary number of memories is straightforward and skipped for brevity.

To handle the memory node, we define a node-marking convention by defining a set $\mathcal{M}$ which contains the marked nodes. We say that a node is marked if it is either itself a memory node, or a node through which a compressed flow is routed.

The modified Dijkstra algorithm starts with finding a node ν closest to node D. Then, we iteratively update the effective distance of the nodes to D. The algorithm is summarized in Algorithm 2. The notation $cost(vD)$, used in Algorithm 2, is in fact the effective distance. At the beginning, $cost(vD) = \infty$ for nodes v not directly connected to D, and then it is calculated for v in every iteration. After finding the effective distance between every pair of vertices via the modified Dijkstra algorithm, we can calculate $\mathcal{F}$ and then $\mathcal{G}$.

Algorithm 2 Modified Dijkstra's Algorithm

```
  $\mathcal{M} = \mu$
  while $V \neq \phi$ do
    $\nu =$ the closest neighbor of $D$.
    $pathlen(\nu) = cost(\nu, D)$
    for $\forall v \in V \setminus \{\nu, D\}$ do
      if $\nu \notin \mathcal{M}$ then
        $cost(vD) = \min\{cost(vD), cost(v\nu) + cost(\nu D)\}$
      else
        $cost(vD) = \min\{cost(vD), \frac{cost(v\nu)}{g} + cost(\nu D)\}$
        $\mathcal{M} \leftarrow \mathcal{M} \cup v$
      end if
    end for
    $V \leftarrow V \setminus \nu$
  end while
```

CHAPTER VI

NETWORK COMPRESSION IN WIRED NETWORKS: INTERNET-LIKE POWER-LAW RANDOM NETWORK GRAPHS

6.1 Introduction

In Chapter 6, the gain of network compression in random power-law graph (RPLG) family is studied. This family is particularly of interest because several studies have shown their resemblance to real-world Internet graphs [4, 34, 22]. Our study entails first finding the optimal strategy for deploying the memory units and then investigating the effect of these memory units on the routing algorithms. The latter is important for the numerical evaluation of the network-wide gain as well. In the following, we first describe the memory deployment problem in RPLG. Then, we discuss the routing problem.

6.1.1 Contributions

Similar to Chapter 4, the gain of network compression in Internet-like power-law graphs is characterized. In particular, through analysis on power-law graphs, it is demonstrated that non-vanishing network-wide gain of memorization is obtained even when the number of memory units is a tiny fraction of the total number of nodes in the network. Furthermore, memory placement in the network poses some challenges to traditional shortest path routing algorithms, as the shortest path is not necessarily minimum cost route in networks with memory. The well-known routing algorithms like Dijkstras algorithm, in their original form are not applicable to networks with memory. This limitation is due to the fact that when the algorithm runs into a memory node on the path, all the previous edges costs along the path should be updated and hence needs to recalculate the whole path. In Chapter 6, the routing problem for compressed flows is solved and a modified Dijkstra's algorithm is presented. Moreover, simulation results are presented which validate the results of the analytical study.

6.2 Memory Deployment in Random Power-law Graphs

Given a set of memory nodes $\boldsymbol{\mu}$, the goal of the memory deployment is to maximize $\mathcal{G}$, by optimally deploying those memories. In [49], authors studied a related problem in the context of caching to maximize the cache hit-rate in the network. There, they showed that the cache placement problem on a general graph is an NP-hard problem. It is straightforward to draw analogies between our memory deployment problem and the cache placement problem, which leads to the fact that the memory deployment problem in a general graph is NP-hard. But, if we limit the deployment problem to certain families of graphs, we can find theoretical solutions that provide insight and enable us to predict the achievable gains.

In order to extend our study to the analysis of the achievable gain $\mathcal{G}(g)$ in general networks, we consider the memory deployment gain in the network graphs that follow the power-law degree distribution [28]. The power-law graphs are particularly of interest because they are one of the useful mathematical abstraction of real-world networks, such as the Internet and social networks. In power-law graphs, the number of vertices whose degree is x, is proportional to $x^{-\beta}$, for some constant $\beta > 1$. For example, the Internet graphs have powers ranging from 2.1 to 2.45 [4, 34, 22]. Accordingly, in the rest of this section we specifically direct our attention to power-law graphs with $2 < \beta < 3$ (which include the models for the Internet graph), and provide results for memory deployment on such network graphs. We wish to study the behavior of the network-wide gain in Internet-like power-law graphs, as a function of the number of the memory units and their locations.

6.2.1 Random Power-law Graph Model

Random Power-law Graph Model: A power-law graph is an undirected, unweighted graph whose degree distribution approximates a power law with parameter β. Basically, β is the growth rate of the degrees. To generate a random graph that has a power-law degree distribution, we consider the Fan-Lu model [28]. In this model, the expected degree of every vertex is given. The Random Power-Law Graph (RPLG), with parameter β, is defined as follows:

Definition of $G(\beta)$ Consider the sequence of the expected degrees $\boldsymbol{w} = \{w_1, w_2, \dots, w_N\}$, and let $\rho = 1/\sum w_i$. For every two vertices v_i and v_j, the edge $v_i v_j$ exists with probability $p_{ij} = w_i w_j \rho$, independent of other edges. If

$$w_i = c i^{-\frac{1}{\beta-1}} \text{ for } i_0 \leq i \leq N + i_0, \tag{37}$$

then graph $G(\beta)$ constructed with such an expected degree sequence is called an RPLG with parameter β. Here, the constant c depends on the average expected degree $\bar{w}$, and i_0 depends on the maximum expected degree Δ. That is,

$$\begin{cases} c &= \frac{\beta-2}{\beta-1}\bar{w}N^{\frac{1}{\beta-1}}, \\ i_0 &= N\left(\frac{\bar{w}(\beta-2)}{\Delta(\beta-1)}\right)^{\beta-1}. \end{cases}$$

With the definition above, it is not hard to show that the expected number of vertices of degree x in $G(\beta)$ is $\approx x^{-\beta}$. In [28], authors showed that for a sufficiently large RPLG, if the expected average degree of $G(\beta)$ is greater than 1, then $G(\beta)$ has a unique giant component (whose size is linear in N), and all components other than the giant component have size at most $O(\log N)$, with high probability. Since we only consider connected networks, we will focus on the giant component of $G(\beta)$ and ignore all sublinear components. Further, by a slight abuse of the notation, by $G(\beta)$ we refer to its giant component. Next, we briefly describe as to how the structure of RPLG provides insight about the efficient placement of memory units.

Although memory deployment problem in a general graph is a hard one, the RPLG with parameter $2 < \beta < 3$ has a certain structure that leads us to finding a very good deployment strategy. The RPLG can be roughly described as a graph with a dense subgraph, referred to as the *core*, while the rest of the graph (called periphery) is composed of tree-like structures attached to the core. Our approach to solve the memory deployment problem is to utilize this property and size the core of $G(\beta)$ and show that almost all the traffic in $G(\beta)$ passes through the core. We propose to equip all the nodes in the core with memory and hence almost all the traffic in $G(\beta)$ would benefit from the memories. The number of memories should be such that the network-wide gain is greater than 1 as $N \to \infty$. Showing that almost all the traffic goes through the core guarantees that $\mathcal{G} > 1$ as shown in Lemma 6.2.2 below. This way, we find an upper bound on the number of memories that should be deployed in an RPLG in order to observe a network-wide gain of network compression. We will also verify that the number of memory units does not have to scale linearly with the size of the network to achieve this gain.

From Definition 6.2.1, we note that the nodes with higher expected degrees are more likely to connect to each other and also other nodes. Therefore, we expect more traffic to pass through these nodes. In our case, we are looking to size the core, i.e., find the number of high degree nodes such that almost all the traffic in the graph passes through them. Theorem 6.2.1 below is our main result regarding the size of the core:

Theorem 6.2.1 *Let $G(\beta)$ be an RPLG. In order to achieve a non-vanishing network-wide gain $\mathcal{G}$, it is sufficient to deploy memories at nodes with expected degrees greater* *than $lw_{\min}$, where l is obtained from*

$$l^{3-\beta} - \frac{1}{\bar{w}\gamma} = 0, \tag{38}$$

and the constant γ is equal to $(1 - \frac{1}{\beta-1})^2 \frac{\beta-1}{3-\beta}$. The set of nodes with expected degree greater than $lw_{\min}$ is defined as core: $\mathcal{C} = \{u | w_u > lw_{\min}\}$.

Proof of the Theorem 6.2.1 follows from the lemmas below.

Lemma 6.2.2 *Let d be the distance between the nodes A and B. Let μ denote a memory unit fixed on the shortest path between A and B, with distance d' from A, i.e., the distance between μ and B is $d - d'$. If the fundamental gain of memory-assisted compression is $g > 1$, then $\mathcal{G} > 1$.*

Proof If there was no memory on the path, we had one unit of flow from A to B and one unit of flow for B to A. Therefore, $\mathcal{F}^0 = 2d$. When memory-assisted compression is performed, the flow going from A to B is reduced to $\frac{d'}{g} + (d - d')$. Similarly, the flow going from B to A is $\frac{d-d'}{g} + d'$. Therefore, $\mathcal{F} = \frac{d'}{g} + (d - d') + \frac{d-d'}{g} + d'$ and thus

$$\mathcal{G}(g) = \frac{2d}{\frac{d'}{g} + (d - d') + \frac{d-d'}{g} + d'} = \frac{2g}{g+1}.$$

Now, considering that $g > 1$, the claim follows. ■

Our approach to find the core is to remove the highest degree nodes from the graph one at a time until the remaining induced subgraph does not form a giant component. In other words, as a result of removing the highest degree nodes, the graph decomposes to a set of *disjoint* islands and hence, we conclude that the communication between those islands must have passed through the core. Therefore, from Lemma 6.2.2, we conclude that in RPLG, we will have a non-vanishing network-wide gain if we choose the core sufficiently big such that the induced periphery of $G(\beta)$ does not have a giant component. The following lemma provides a sufficient condition for not having a giant component in RPLG.

Lemma 6.2.3 ([28]) *A random graph $G(\beta)$ with the expected degrees $\boldsymbol{w}$, almost surely has no giant components if*

$$\frac{\sum_i w_i^2}{\sum_i w_i} < 1. \tag{39}$$

Lemma 6.2.4 *Consider a random graph G with the sequence of the expected degrees $\boldsymbol{w}$. If U is a subset of vertices of G, the induced subgraph of U is a random graph with the sequence of the expected degrees $\boldsymbol{w'}$, where*

$$w_i' = w_i \frac{\sum_{v \in U} w_v}{\sum_{v \in G} w_v}.$$

Proof The probability that an edge exists between two vertices of U is equal to the edge connection probability in G. Consider a vertex u in U. The expected degree of u is

$$\rho \sum_{v \in U} w_u w_v = w_u \frac{\sum_{v \in U} w_v}{\sum_{v \in G} w_v}.$$

■

Proof of Theorem 6.2.1 Consider a $G(\beta)$ with the set of lowest degree nodes U_l, all having expected degrees in the interval $(w_{\min}, l w_{\min})$. According to Lemma 6.2.3, to ensure that the induced subgraph G_{U_l} does not have a giant component, we should have $\sum_{v \in U_l} {w_v'}^2 / \sum_{v \in U_l} w_v' < 1$, where $w_v' = w_v \frac{\sum_{v \in U_l} w_v}{N \bar{w}}$ as in Lemma 6.2.4. To find w', we should first obtain $\sum_{v \in U_l} w_v$. According to [28], we have

$$\begin{aligned} \textstyle\sum_{v \in U_l} w_v &\approx N\bar{w}(1 - l^{2-\beta}), \\ \textstyle\sum_{v \in U_l} {w_v}^2 &\approx N\bar{w}^2 (1 - \tfrac{1}{\beta-1})^2 \tfrac{\beta-1}{3-\beta} l^{3-\beta}. \end{aligned} \tag{40}$$

From (40), we conclude that $w'_v = (1 - l^{2-\beta})w_v$, for all $v \in U_l$. Thus we have,

$$\sum_{v \in U_l} w'_v \approx N\bar{w}(1 - l^{2-\beta})^2. \tag{41}$$

Similarly,

$$\sum_{v \in U_l} {w'_v}^2 \approx N\bar{w}^2 \gamma l^{3-\beta}(1 - l^{2-\beta})^2. \tag{42}$$

Combining (41) and (42), we obtain the relation in (38) between β and l. Having l, we can easily obtain the size of U_l by finding the number of vertices with expected degree less than $lw_{\min}$ which is readily available from (37). ■

Theorem 6.2.1 provides the required information to find the size of the core and hence the number of memory units. As finding the closed-form solution for the size of the core is not straightforward, we use numerical analysis to characterize the number of required memory units using the results developed above.

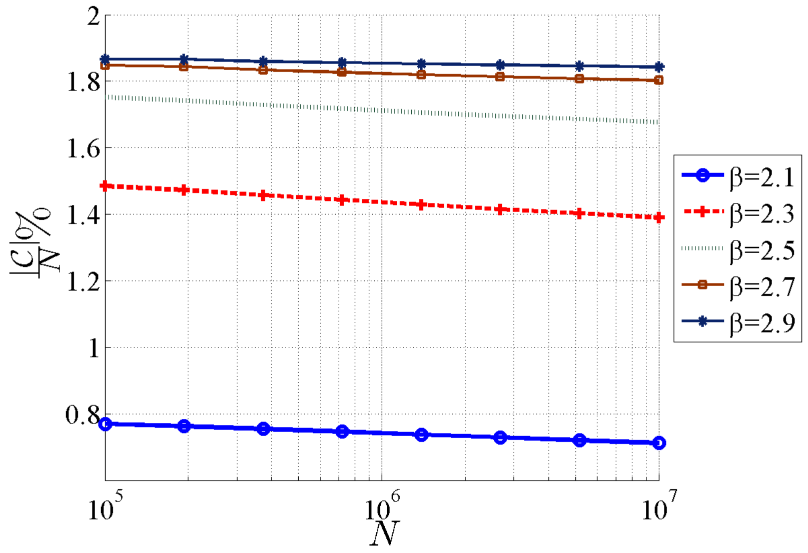


Figure 22: The scaling of the core size $\frac{|\mathcal{C}|}{N} \times 100$ versus N for different β's.

In Figure 22 the scaling of the core size versus N is depicted for various β's. As we see, the core size is a tiny fraction of the total number of nodes in the network and this fraction decreases as N grows. This is a promising result as it suggests that by deploying very few memory units, we can reduce the total amount of traffic in a huge network.

6.3 Simulation Results

To validate our theoretical results, we have conducted different sets of experiments to characterize the network-wide gain of memory in RPLG. For experiments, we used DIGG RPLG generator [21], with which we generated random power-law graph instances with number of vertices between 1000 and 5000, and $2 < \beta < 3$. The result are averaged over 5 instances of generated RPLG. In our simulations, we report results for various core sizes (number of memory units).

We first verify our assumption that a tiny fraction of the highest degree nodes observes most of the traffic in the network. Figure 23 shows the fraction of the paths that pass through the core (FPPC) for different core sizes and β's. As we expected, more that 90% of the shortest paths in the graph involves less than 2% of the highest degree nodes to route the flow. Although our theoretical result in Theorem 6.2.1 is asymptotic in N, Figure 23 suggests that our result holds for moderate values of N as well. Therefore, we can place the memory units at the core and results can be extrapolated for large graphs with large number of nodes.

To validate the network-wide gain of memory, we have considered two RPLGs with sizes $N = 2000$ and $N = 4000$. Assume that each memory node has observed a sequence of length $m = 4$MB of previous communications in the network. The packets transmitted in the network are of size 1kB. This assumption is in accordance with the maximum transmission unit (MTU) of 1500-bytes allowed by Ethernet at the network layer. From our results in Chapter 2, we observe that a memory-assisted source compression gain of $g \approx 2.5$ is achievable for real traffic traces. Hence, we use $g = 3$ in our simulations.

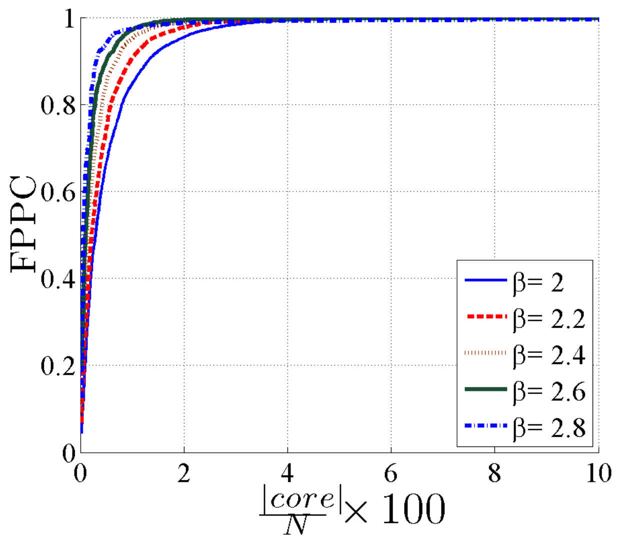


Figure 23: The fraction of the paths passing through the core (FPPC) vs. the core size, for RPLG of size $N = 5000$.

To verify the results of routing with memory in Chapter 5, we conduct the following experiment. If we do not use the modified Dijkstra's algorithm in the networks with memory (i.e., we do not optimize the routing algorithm to utilize the memories), as Lemma 6.2.2 suggests, the network-wide gain would be bounded by $\frac{2g}{g+1}$. Therefore, even for very large values of g, the network-wide gain would remain less than two (as shown in Figure 24), which is not desirable. Figure 25 describes our results for the achievable network-wide gain of memory-assisted compression. We measured the total flow without memory. We also obtain the optimal paths when we have memory units are deployed. We consider three cases in which the fraction pf nodes equipped with memory increases from 2.5% of the nodes to 10%. All data has been averaged over the 5 graphs in each set.

The trendlines suggest that $\mathcal{G}$ increases as β increases which is expected since the FPPC increases with β. In other words, more traffic between the nodes in periphery has to travel through the dense subgraph (core) as β increases. Further, by increasing the number of memory units, the network-wide gain increases and approaches to the upper bound g. It is important to note that enabling only 2.5% of the nodes in the network with memory-assisted compression capability, we can reduce the total traffic in the network by a factor of 2 on top of flow compression without using memory, i.e., end-to-end compression. We emphasize that this memory-assisted compression (UcompM) feature does not have extra computation overhead for the source node (in comparison with the end-to-end compression technique in Ucomp). Further, this feature only requires extra computation at the memory units when compared to Ucomp. However, this extra computation incurs a linear complexity with the length of the data traffic. Hence, overall with some additional linear computational complexity on top of what could have been achieved using a mere end-to-end compression, the memory-assisted compression can reduce the traffic by a factor of 2.

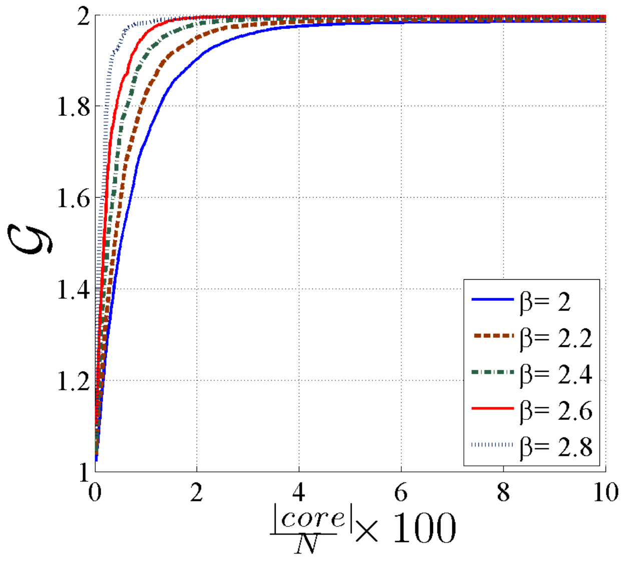
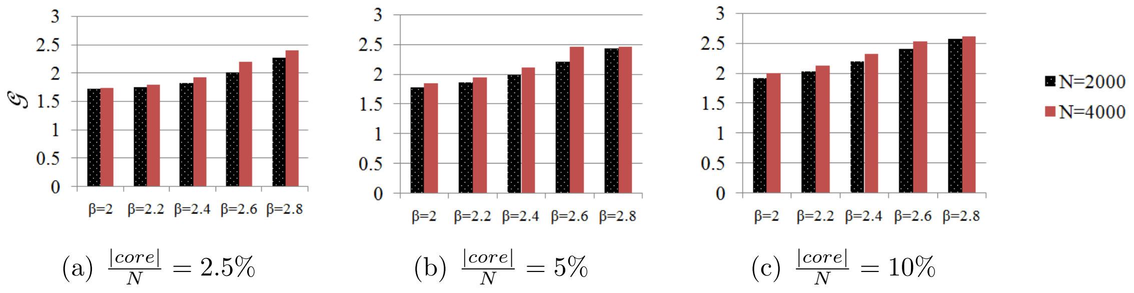


Figure 24: Illustration of the network-wide gain when simple Dijkstra routing is used. Note that $\mathcal{G}$ is capped.



(a) $\frac{|core|}{N} = 2.5\%$ (b) $\frac{|core|}{N} = 5\%$ (c) $\frac{|core|}{N} = 10\%$

Figure 25: Network-wide gain of memory assisted compression $\mathcal{G}$ for different core sizes and power-law parameter β, for $g = 3$.

CHAPTER VII

WIRELESS NETWORK COMPRESSION VIA MEMORY-ENABLED OVERHEARING HELPERS

7.1 Introduction

Mobile data efficiency is an important feature of wireless communication. It increasingly draws attention as providers face the difficulty of handling the explosive increase in the demand and look for solutions to reduce the cost of data delivery in wireless networks. One potential solution is to find ways to eliminate the redundant data that is being transmitted to clients through the bottle-neck of the network, the most important being the last hop: the wireless link from the wireless gateway to the mobile client. As suggested in the following, there are two main dimensions that contribute to the redundancy within a network. 1) redundancy within the content; and 2) redundancy across different clients. IP-layer Redundancy Elimination (RE), in the form of repetition suppression for a single client, has been successful on the last hop links; In [52], based on the data gathered from gateways in North American and European wireless service providers, authors show that most mobile users can save bandwidth with RE, some reduce their traffic volume by as much as 50%. In another study [68], based on data traces collected from both laptop and smartphone users over a span of three months, authors show that an average of 20% redundancy exists within each users's data trace. These saving are obtained in the first dimension of redundancy. Further, recent studies in [41] show that traces derived from real-world wireless traffic collected in a noise-free environment contain around 50% inter-client repetition within packets, i.e., duplicate strings across packets. All these signify the importance of redundancy elimination in the wireless flows. However, all the existing works [52, 68, 41] confine themselves to deduplication of repeated patterns for redundancy elimination. While deduplication is effective in removing long repeated bit sequences, deduplication is not suitable for suppressing sub-packet level statistical dependencies in the data that are not mere repeatitions. It is expected to remove an even more substantial amount of redundancy using information-theoretic compression methods when memory-enabled nodes are present in the network. The proposed wireless network compression via memory-enabled overhearing helpers is focused on these information-theoretic methods.

In Chapter 2 and Chapter 3, we took the first steps towards characterizing the achievable benefits of exploiting the packet redundancies beyond simple repetition suppression (i.e., de-duplication). Data compression and source coding are natural candidates for this task. However, traditional compression techniques would be ineffective in the elimination of the aforementioned two types of redundancies. The reasons are the following: 1) redundancy within a packet cannot be effectively removed due to small size of the packet [12], and 2) traditional compression methods cannot leverage the redundancy across clients; as they compress each packet independent of the other packets. In [71, 70], we formulated the redundancy elimination as *network compression via network memory* and introduced a new framework for compression of network data called *memory-assisted compression.* This approach departs from the traditional source coding techniques in that it relies on the network memory for compression. We have already explored the network-wide gain of memory-assisted compression in wired networks. However, the gain of memorization and memory-assisted compression is more spelled out in the bandwidth-constrained wireless networks. On the other hand, it is more challenging to establish the memorization scheme in such networks and our solution for wired networks [71, 70] is not directly applicable to the last hop in wireless networks. This is mainly because the broadcast nature of the wireless networks and also the asymmetric cost of transmission in different links that need to be incorporated in the network compression framework while guaranteeing that all packets are recoverable in a strictly lossless manner at the clients.

7.1.1 Contributions

In Chapter 7, we study wireless network packet compression via memory-enabled overhearing helpers. In particular, we explore the benefits of *two-part coding with asymmetric cost* in the last hop wireless links, from the wireless gateway to the mobile clients, by deploying memory-enabled helpers with much less costly links to the mobile clients [72, 69]. The memory-enabled helpers are small, possibly cooperative nodes with no backhaul connectivity but with sufficiently large storage space. The overhearing capability of helper nodes that comes at no cost eliminates the need for backhaul connectivity in our work. The helper nodes bring throughput enhancement and gateway off-loading which would be of particular interest for WiFi and cellular networks. The memory-enabled helpers overhear previously transmitted packets from the wireless gateway to mobile clients. These overhearing packets provide statistical information about the traffic. Then, in the compression of a new packet, this information is sent from the overhearing (helper) node to a mobile client to supplement (as a side information) the compressed data from the wireless gateway to the mobile client; enabling the client to decompress the codeword and recover the packet. Since the communication in the link between the overhearing memory-enabled helper and the client is by far less costly than that of the wireless gateway and the client, the proposed network compression via overhearing nodes, by design reduces traffic on the link from the wireless gateway to the mobile client. In Chapter 7, we study both analytically and experimentally the fundamental limits of the wireless network compression via overhearing (memory-enabled) nodes. We stress that our scheme is universal in the sense that we do not assume to know the distribution of the source traffic a priori.

The network compression (via helpers) proposed in this chapter overcomes the shortcomings of traditional universal compression. The use of helper nodes in wireless networks is studied in various contexts from femto-cell network architectures [26] to device-to-device collaboration in wireless networks [37]. The use of helper nodes in the network in this work is inspired by these designs.

In Section 7.2, the application of memory-assisted compression for a sample wireless network scenario is presented and an abstraction of the compression problem is described. Furthermore, a practical code design and the performance analysis of the code is presented. Simulation results, performed in NS-2 simulator [58], are provided in Section 7.6. Moreover, we provide a discussion about the impact of loss on network compression in Sec. 7.5.

7.2 Redundancy Elimination in Wireless Networks via Memory-Assisted Compression

Consider Figure 26 for an example scenario involving a single wireless gateway S, a mobile client C and a helper M. The idea is to deploy memory-enabled helpers that are capable of overhearing communication from the wireless gateway to all the mobile clients inside the coverage area of the wireless gateway. The overhearing comes at no extra cost due to the broadcast nature of the wireless communication. Although this can be applied to every cellular or WiFi access networks, one realization of such memory-enabled helpers can be in femto-cell network designs combined with traditional macro-cell networks, as in [26]. Here, we note that the backbone connectivity of the helpers are not included in the problem setup, first because the learning process of helper nodes is performed only based on the overheard data which is available for free and secondly, solutions that rely on helper connectivity should include provisions to deal with intermittent connectivity of the backbone connection and also the effect of extra load on the backbone connection should be justified.

The proposed network compression via memory at the overhearing nodes works as follows. First, recall that the traffic (i.e., the packets) destined to different mobile clients from the gateway S are highly correlated. Therefore, the overhearing memory-enabled helpers can overhear the past communication between the cell tower (or the WiFi access-point) and mobile nodes and hence, learn the statistical properties of the packets in the traffic. These extracted statistical properties can then be used as a side information (if provided to the client) improving the compression performance on the future traffic from the gateway S to any mobile client. In other words, the memory-enabled helpers can possibly help to reduce the transmission load of the cell tower by transmitting the side-information about the data traffic to the clients using a *less costly* memory-client M-C link.

Note that the main objective of network compression is to minimize the total communication cost and hence support more clients. As such, we can define a virtual cost for S-C and M-C links in Figure 26. Let κ denote the ratio of the cost of communicating one bit in the M-C link to that of the S-C link. In practical settings, it is rational to assume that the S-C link is much more costly than the M-C link. Hence, κ is much smaller than unity. We use the parameter κ in our analytical development to minimize the aggregate cost of communication. This departs from the objective of only minimizing the total number of bits transmitted from a source to a destination in a traditional setup. It is imperative to note that the memory-assisted compression via overhearing memory-enabled helpers would provide additional compression benefits over and beyond those already gained by traditional (end-to-end) compression techniques, i.e., compressing the packet from S to C while there is no memory deployed. Further, the proposed network compression only entails negligible extra computational overhead at the wireless gateway and the overhearing memory-enabled helper while reducing the aggregate cost.

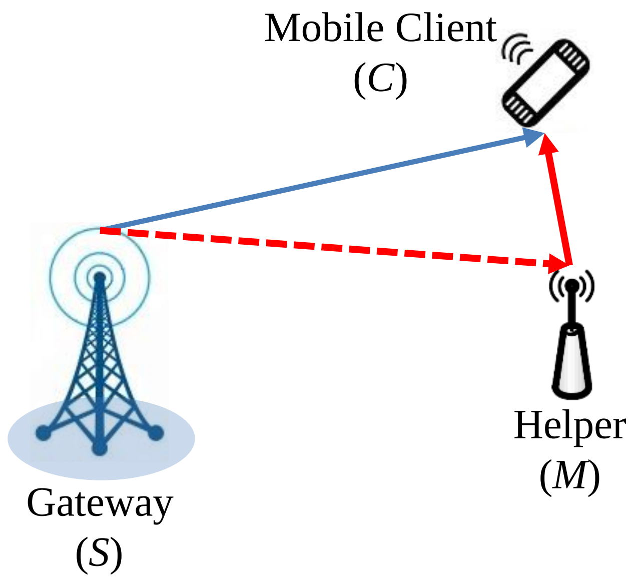


Figure 26: An illustrative example of a wireless network with a single helper (deployed memory-enabled helper). A short-lived connection to the source by a mobile client is shown by a solid arrow. Overhearing is shown by a dashed arrow. The link supplementing side-information is shown by a thick solid arrow.

7.2.1 Setup

The abstract model of the network compression via overhearing memory-enabled helper is shown in Figure 27, for a single helper and a client. We consider the traffic reduction (compression) over the down-link. The data are delivered from the wireless gateway S, which is the source in our abstraction, to the mobile client C. We only consider schemes where the sequence x^n, i.e. the packet, is exactly recoverable at the client. Therefore, all the compression schemes considered are strictly lossless.

Definition Let $\mathcal{A}^n$ be the set of all sequences of length n over alphabet $\mathcal{A}$. The code $c_n(\cdot) : \mathcal{A}^n \to \{0,1\}^*$ is called strictly lossless if there exists a reverse mapping $d_n(\cdot) : \{0,1\}^* \to \mathcal{A}^n$ such that

$$\forall x^n \in \mathcal{A}^n : \quad d_n(c_n(x^n)) = x^n.$$

All of the practical data compression schemes are examples of strictly lossless codes, namely, the arithmetic coding and Huffman coding.

The source is assumed to generate and send different packets to mobile nodes one at a time (unicast). The memory-enabled helpers are assumed to overhear the communication from S to client C. Each overhearing memory-enabled helper is also assumed to be capable of sending information to those mobile nodes in its vicinity.

The link between S and C is lossy due to the wireless channel but we assume a proper feedback for packet retransmission would take care of packet losses on the S-C link. Note that in practice, we consider a stable situation which is when the transitional memorization phase is over. In other words, we assume that every overhearing node has been in the network for a long time and has accumulated sufficient knowledge about the source contents' model from all the past communication (the memorization phase). In practice, it is rational to assume that the memory-enabled helper has observed a sufficient number of packets (when S was serving several other clients), and hence, the total size m of memorized packets is assumed to be sufficiently large. This assumption is not necessary for network compression but it would simplify our presentation. For example, in the experiment plotted in Figure 4(a), we observed that $m = 4$MB is sufficient memorization and having further memory has subtle effect on the gain of compression.

Figure 27 shows the abstraction of the setup in Figure 26. The basic principle in network compression in Figure 27 can be described as following. Assume that both M and S share the memory y^m after the memorization phase. Further, assume we use a two–part statistical compression method. A model of the source S is created at S and M using the memory y^m. Then, in the compression of a new packet x^n, the server S would only send (to the client C) the output of the arithmetic encoder which compresses x^n using the model. To complement the compressed sequence sent by S, the memory-enabled helper M forwards (to the mobile client C) the corresponding source model used by arithmetic encoder. Hence, the mobile client would be able to decode x^n although the client did not have memory (i.e., the source model). In this scenario, the cost of sending a compressed packet using memory (on the link S-C) would be very low relative to the cost of sending the source model on the link from the memory-enabled helper to the client. This would achieve the principle objective of network compression which is saving the cost on the link from the wireless gateway S to the mobile client.

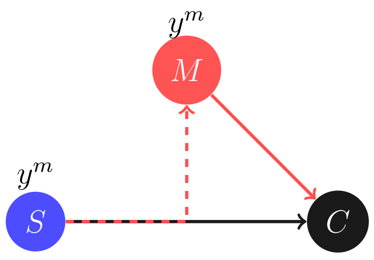


Figure 27: The abstract illustration of the traffic reduction problem via network compression. The memorized sequence y^m represents the total past data overheard by M from S to the clients.

As mentioned before, since we wish to reduce the load of the gateway, we have an asymmetric situation where a higher cost is associated with the channel from the source to the client than from the helper to the client. This asymmetry between the channel costs is motivated by real-world cellular networks where a single base-station serves a large number of clients. Hence, if the load of the base-station by each client is reduced, it can potentially serve a larger number of clients. For example, the S-C link from the base-station to the client (and hence the overhearing link S-M) can operate in a frequency different from the M-C link. Whenever the base-station hands-off the connection to the mobile client (and the overhearing memory-enabled helper), its frequency slot frees up and a new client can be served. Further, due to a lower communication radius, the frequency slot allocated to the M-C link can be reused within a cell for the link between some other memory-enabled helpers with nearby clients. This architecture together with the proposed network compression offers a significant opportunity for traffic reduction so as to deliver x^n by exploiting the side-information y^m shared between S and M.

As shown in Figure 27, the memorized sequence y^m is already available at both S and M. Let x^n be a packet of length n to be delivered from the source to C. The problem of interest, in its general form, is as to how the encoder of S would encode x^n such that the aggregate communication cost on the link S-C together with the cost of supplemented bits on the link M-C would be minimized, provided that x^n would be recovered at the client. This general formulation, would reduce to the objective of minimizing the load of the gateway if we further assume that the cost of the transmission on M-C link is negligible.

Table 4: Summary of wireless network compression via two–part codes.

Initialization (S and M)
The helper node M overhears the communication of S with past clients and accumulates knowledge about the source model and its statistics. Node S also performs the same operations to construct the model. The total sequence size observed by M is y^m.
Operation (S)
For every new packet (sequence) x^n, S uses the statistical model to estimate the probability of the symbols in x^n. Then, these probabilities estimates along with x^n are sent to an encoder (e.g., an arithmetic encoder). The output of the encoder (NOT the probability estimates) is then sent to C.
Operation (M)
Once the helper M finds out that the compressed packet $c(x^n)$ is sent to a client within its coverage, then M sends the probability estimates necessary for decompression to C.
Operation (C)
The client C receives the output of the (arithmetic) encoder from S and the probability estimates from M and feeds them to a decoder (e.g., arithmetic decoder) to reconstruct x^n.

7.3 Code Design for Network Compression via Overhearing Helper

The main feature of our approach is that the source can rely on the memory-enabled helper to send side-information to clients. In the statistical universal compression technique, this side information is in fact the source model formed at the overhearing memory-enabled helper using the sequence y^m. Now, the question is how much this side information can improve the efficiency of compression at the source. Further, which coding mechanism would realize that gain. We first review the traditional two–part coding scheme (c.f. [38, 12] and the references therein) and introduce the necessary notations and describe its adaptation for our network compression via overhearing memory-enabled helpers.

7.3.1 Traditional Two–part Code

For the analysis, we assume that S is a stationary parametric source. Let μ_θ be the probability density function of the source depending on a d-dimensional parametric vector θ which takes values in $\Theta \subset \Re^d$, where $\Re$ is the set of real numbers. Consider a parametric source with probability density function μ_θ. By this setup, for example, for a binary Bernoulli (memoryless) source which is represented with a single parameter γ, the probability that the source would output the sequence x^n with k ones and $n-k$ zeroes is given by $\mu_\gamma(x^n) = \gamma^k(1-\gamma)^{n-k}$. Please see [38] for a more detailed discussion of the parametric sources.

If the parameter vector $\theta \in \Theta$ was known, the ideal code length of a packet x^n, obtained from the Shannon code, would be $\log 1/\mu_\theta(x^n)$ [32]. On the other hand, since in practice the parameter θ is not known a priori, we wish to encode the packet using a universal probability distribution $P(x^n)$, which is in some sense close to the true unknown probability distribution. The universal two–part code [12, 11, 64] provides a practical solution to this problem with close to optimal code lengths which can be simply implemented.

The traditional (without memory) two–part source coding scheme encodes a packet x^n as follows: the first part of the code basically describes the best estimation $\hat{\theta}(x^n) \in \Theta$ of the unknown source parameter vector $\theta^\star$ by using the statistics of the source extracted from the packet x^n. This estimate $\hat{\theta}(x^n)$ is then used in the second part for compression of the packet x^n. Therefore, roughly speaking the closer the estimate $\hat{\theta}(x^n)$ gets to the true parameter $\theta^\star$, the smaller the description length of the packet to be compressed becomes (i.e., better compression is achieved on the average). However, in order to provide a better estimate for $\theta^\star$, we would need more bits to describe $\hat{\theta}(x^n)$. To achieve the best code length, one should use an estimate of the source parameter that minimizes the total code length of the two–part code, which is the sum of the lengths of the two parts of the code. Let $l^{2p}(x^n)$ be the length of the codeword assigned to x^n by the two–part encoder. We have

$$l^{2p}(x^n) = \min_{\hat{\theta}(x^n)} \{l(\hat{\theta}(x^n)) - \log P_{\hat{\theta}(x^n)}(x^n)\}, \tag{43}$$

where, $l(\hat{\theta}(x^n))$ is the universal length of the codeword describing the estimate $\hat{\theta}(x^n)$ and $-\log P_{\hat{\theta}}(x^n)$ is the description length of a packet x^n given the estimate $\hat{\theta}$ when the optimal Shannon code is used.

The optimization in (43) is studied in the literature and performance of two–part codes has been characterized (cf. [38, 10, 66, 12]). In short, the code lengths of traditional two–part code are characterized by the following theorem:

Theorem 7.3.1 *For a d-dimensional parametric source with parameter $\theta \in \Theta$, we have*

$$\begin{aligned}\mathbf{E}[l^{2p}(X^n)] &= H_n(\theta) + \frac{d}{2}\log n + \\ &\quad \log \int_{\lambda\in\Theta} |\mathcal{I}(\lambda)|^{\frac{1}{2}}\, d\lambda + O(1),\end{aligned} \tag{44}$$

where $|\mathcal{I}(\cdot)|$ denotes the determinant of the Fisher information matrix, defined in (50).

7.3.2 Two–part Code with Asymmetric Cost

The two–part coding strategy fits well within the framework of network compression in Figure 27. The benefit of using two-part coding strategy for wireless network compression problem is two-fold. First, the compressed codeword describing x^n is consisted of two parts that can be separately sent to the end-user, i.e., the client node; one part from the source and the other from the memory-enabled helper, as shown in Figure 27. We will incorporate the asymmetric cost of communication of the two part of the code in our analysis. Secondly, the memorized sequence y^m that is longer than x^n can be used for obtaining a better estimate of the source parameter which translates to more efficient communication.

To proceed we will consider the natural asymmetric cost of transmission that happens in our setup since the cost of transmission from the source to the client is significantly higher than that of the link between the memory-enabled helper and the client.

From (43), $l^{2p}(x^n)$ is the length of the sequence sent by the source when the source compresses the packet x^n without regard to the memory-enabled helper. On the other hand, by exploiting the memory-enabled helper and hence obtaining the estimate $\hat{\theta}(y^m)$, the source only needs to send the second part of the two–part description, i.e., $-\log P_{\hat{\theta}(y^m)}(x^n)$. The client receive this codeword, however, at this point it is unable to decode the codeword as the source parameter used in encoding is unknown to the client. Thus, the codeword corresponding to $\hat{\theta}(y^m)$ is transmitted by the memory-enabled helper M to the client. This codeword has length $l(\hat{\theta}(y^m))$.

Therefore, the total cost of delivering x^n in Fig 27, denoted by $\mathcal{C}(x^n)$, is given by

$$\mathcal{C}(x^n) = \min_{\hat{\theta}(y^m)} \; \{\kappa l(\hat{\theta}(y^m)) - \log P_{\hat{\theta}(y^m)}(x^n)\}, \tag{45}$$

where $\kappa \ll 1$ is ratio of the cost of transmission on the M-C link to that of the S-C link.

To characterize the average cost of communication between the source and the destination, we need to find the expected value of (45) with respect to the true source parameter, i.e., $\mathbf{E}_{X^n \sim \theta^\star}[\mathcal{C}(X^n)]$. In the next section, we investigate the trade-offs of the code design and characterize the expected communication cost.

7.4 Performance Evaluation of Network Compression via Overhearing Helper

The key to constructing a two–part code achieving the minimum communication cost is to discretize Θ to a countable set of points $\Phi \subset \Theta$ such that the ML estimator restricted to Φ achieves almost the same codelength as the unrestricted ML estimator [38]. The finer the discretization, the smaller the number of bits in the S-C communication link would be. But, finer descritization would result in larger codelengths transmitted from M to C. We are after the optimal trade-off between these two parts minimizing the total communication cost.

Denote the entropy of the parametric source at S generating x^n by $H_n(\theta)$, defined as

$$H_n(\theta) = \sum_{x^n} \mu_\theta(x^n) \log \frac{1}{\mu_\theta(x^n)} = \mathbf{E}\left[\log \frac{1}{\mu_\theta(X^n)}\right]. \tag{46}$$

The following theorem determines the communication cost in the case of network compression via overhearing helper. Let y^m be a sequence available at S and M generated by a d-dimensional parametric source. A sequence of length n is to be transmitted from S to C. Let $\mathcal{L}_S$ be the expected number of bits sent by S and $\mathcal{L}_M$ be the expected number of bits sent by M to the client.

Theorem 7.4.1 *Given a memory of size m such that $\frac{n}{m} = o(1)$,*[1] *we have*

$$\begin{cases} \mathcal{L}_S &= H_n(\theta) + o(1) \\ \mathcal{L}_M &= \frac{d}{2}\log m + \int_{\lambda\in\Theta} |\mathcal{I}(\lambda)|^{\frac{1}{2}} d\lambda + O(1) \end{cases},$$

where $|\mathcal{I}(\cdot)|$ denotes the determinant of the Fisher information matrix.

Proof Proof is provided in the Section 7.4.1. ■

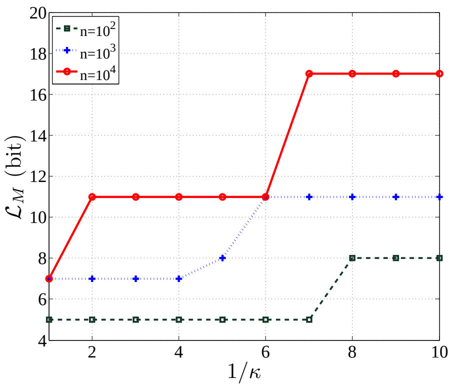
Corollary 7.4.2 *Let κ be the ratio of the cost of transmission on the M-C link to that of S-C link. The expected communication cost in Figure 27 is*

$$\mathbf{E}[\mathcal{C}(X^n)] = H_n(\theta) + \kappa \left[\frac{d}{2}\log m + \int_{\lambda\in\Theta} |I(\lambda)|^{\frac{1}{2}} d\lambda + O(1)\right]. \quad (47)$$

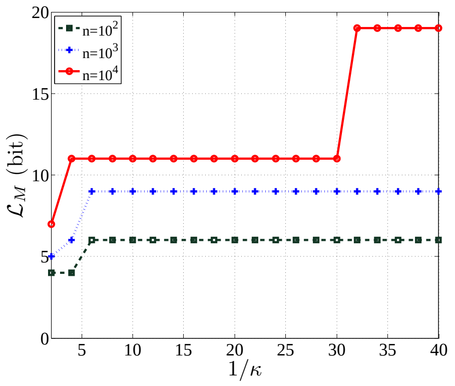
Remark: Theorem 7.4.1 demonstrates that when m is sufficiently large, the output of the source is close to the sequence entropy which is the information-theoretic lower bound on the source output size. From corollary 7.4.2, we observe that when we have an asymmetry in communication cost, i.e., $\kappa \ll 1$, larger memorized sequences can be employed while the total communication cost is still dominated by the bits sent from the server to the client.

Example: To illustrate the trade-offs in Theorem 7.4.1, we consider a memoryless source model with alphabet size 256, i.e., each symbol of the source is 1 byte. For small packet lengths, a memoryless source model suffices for modeling of the underlying source in practice so as to avoid overfitting [73]. For simplicity, we uniformly discretize the parameter space. The memorized sequence is used to choose the best source parameter from the discretized space. The discretization is done such that the sum of $\mathcal{L}_S$ and $\mathcal{L}_M$ is minimized. The length n packet to be compressed together with the estimated parameter is then fed to a standard arithmetic coder [50].

[1] $f(n) = o(g(n))$ if and only if $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.



(a)



(b)

Figure 28: Illustration of the optimal number of bits $\mathcal{L}_M$ that the memory node needs to send to the client versus the relative cost of links for different sequence lengths n and the alphabet size 2: (a) memoryless source and (b) order-1 Markov source.

In Figure 28, the optimal $\mathcal{L}_M$ for various cost ratio is depicted for both a memoryless parametric source (Figure 28(a)) and an order-1 Markov source (Figure 28(b)). As the cost ratio κ assumes smaller values, the optimal number of bits $\mathcal{L}_M$ that the memory node sends to the client increases.

Figure 29 shows the ratio $\frac{\mathcal{L}_M}{\mathcal{L}_S}$. For example, for a packet length of 1kB, the size of the parameter estimate is roughly the same size as the compressed packet using the aforementioned arithmetic coder, i.e., $\frac{\mathcal{L}_M}{\mathcal{L}_S} \approx 1$. To quantify the gain of network

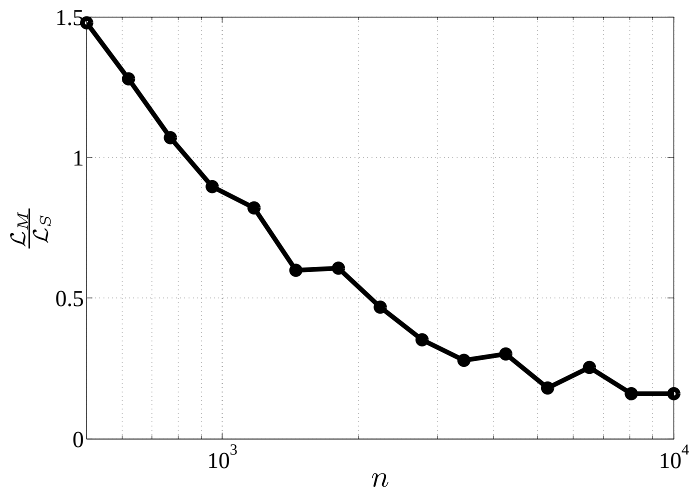


Figure 29: Ratio of the output size of the helper $\mathcal{L}_M$ to the source output size $\mathcal{L}_S$ vs. the packet size n for a memoryless source with an alphabet size 256.

compression with memory-enabled helpers over traditional compression, we define

$$g = \frac{\mathbf{E}[l^{2p}(X^n)]}{\mathbf{E}[\mathcal{C}(X^n)]}.$$

For packet sizes of length 1kB where $\frac{\mathcal{L}_M}{\mathcal{L}_S} \approx 1$, when $\kappa \ll 1$, we observe that $g \approx 2$, as the communication cost of M-C link is negligible.

The result of Figure 29 is used in the simulations of Section 7.6 to determine the output bit rate of the source and the memory-enabled helper to the client.

7.4.1 Proof

Consider a d-dimensional parameter space Θ where the true source parameter $\theta^\star$ is chosen from. Further, let the parameter $\theta^\star$ be chosen according to a prior density $w(\theta)$ defined over Θ. The two–part coding with memory y^m is comprised of three steps. First, the ML estimate of $\theta^\star$ is obtained from the memorized sequence; this estimate is denoted by $\hat{\theta}(y^m)$. In the second step, to find a codeword describing the ML estimate the space Θ is split into a set of regions $\mathbf{R}$; the center point of each region $R \subset \mathbf{R}$, denoted by ϕ_R, is used to discretize Θ. Let

$$\Phi = \bigcup_R \{\phi_R\}$$

be the discretized space and denote the corresponding ML estimate in Φ closest to $\hat{\theta}(y^m) \in R$ by $\hat{\phi}_R(y^m)$.[2] Let $\omega(\phi_R)$ denote the probability density corresponding to $w(\theta)$ in the discretized space Φ. We have

$$\omega(\phi_R) = \int_{\theta \in R} w(\theta) \, \mathrm{d}\theta.$$

Finally, a sequence x^n is compressed using a Shannon code (which is the optimal code when the source parameter vector is known) with parameter vector $\hat{\phi}(y^m)$. The description length of x^n using the Shannon code is given by $-\log P_{\hat{\phi}(y^m)}(x^n)$. The description length of the parameter $\hat{\phi}(y^m)$ is $-\log \omega(\hat{\phi}(y^m))$. Please note that the Shannon code is sent by S whilst the parameter is transmitted from M which is less costly by a factor $\kappa \ll 1$. Henceforth, the communication cost of transmitting x^n from source to client with memorized sequence y^m available to the memory-enabled helper can be written as

$$\mathcal{C}(x^n) = \log \frac{1}{P_{\hat{\phi}(y^m)}(x^n)} + \kappa \log \frac{1}{\omega(\hat{\phi}(y^m))}. \tag{48}$$

By adding and subtracting a $-\log \mu_{\theta^\star}(x^n)$ term (which is the length of the Shannon code using the true source parameter)and a $-\log P_{\hat{\theta}(y^m)}(x^n)$ term (which is the length of the Shannon code using the maximum likelihood parameter after y^m is observed) from (48) and then taking expectation, we can write the expected communication

[2]The subscript R is dropped when it is clear from the context.

cost as in (49).

$$\begin{aligned}
\mathbf{E}_{X^n}[\mathcal{C}(X^n)] &= -\kappa \log \omega(\hat{\phi}(y^m)) + \mathbf{E}_{X^n}\left[\log \frac{1}{P_{\hat{\phi}(y^m)}(X^n)}\right] \\
&= -\kappa \log \omega(\hat{\phi}(y^m)) + \mathbf{E}_{X^n}\left[\log \frac{P_{\hat{\theta}(y^m)}(X^n)}{P_{\hat{\phi}(y^m)}(X^n)}\right] + \\
&\quad + \mathbf{E}_{X^n}\left[\log \frac{\mu_{\theta^\star}(X^n)}{P_{\hat{\theta}(y^m)}(X^n)}\right] + \mathbf{E}_{X^n}\left[\log \frac{1}{\mu_{\theta^\star}(X^n)}\right]. \qquad (49)
\end{aligned}$$

Proof of Theorem 7.4.1 follows from the following lemmas that examine the terms in (49). The lemmas are adaptation of our previous results on the performance of two–part codes from [12] and other results in the literature [38]. The rightmost term in (49) is the easiest to evaluate. By definition (46), we have $\mathbf{E}_{X^n}\left[\log \frac{1}{\mu_{\theta^\star}(X^n)}\right] = H_n(\theta^\star)$. For the second term from the right, we have the following lemma.

Lemma 7.4.3

$$\mathbf{E}_{X^n}\left[\log \frac{\mu_{\theta^\star}(X^n)}{P_{\hat{\theta}(y^m)}(X^n)}\right] = c_1 \frac{n}{m} + O\left(\frac{1}{\sqrt{m}}\right),$$

where $c_1 > 0$ is a constant.

Proof Consider the Taylor's expansion of the term $-\log \mu_{\theta^\star}(X^n)$ around the ML estimate. We have

$$\begin{aligned}
\mathbf{E}_{X^n}\left[-\log \mu_\theta(X^n)\right] &= \mathbf{E}_{X^n}\left[-\log P_{\hat{\theta}(y^m)}(X^n)\right] + \\
&\quad \left(\nabla \log \frac{1}{\mu_\theta(X^n)}\right)(\theta - \hat{\theta}(y^m)) + \\
&\quad \frac{n}{2}(\theta - \hat{\theta}(y^m))^{\mathrm{T}} \mathcal{I}(\hat{\theta})(\theta - \hat{\theta}(y^m)) + \\
&\quad O\left(\frac{1}{\sqrt{m}}\right),
\end{aligned}$$

where $\mathcal{I}(\hat{\theta})$ is the expected Fisher information matrix evaluated at $\hat{\theta}$, defined as

$$\mathcal{I}_{ij}(\hat{\theta}) = \mathbf{E}_{X^n}\left[-\frac{\partial^2}{\partial\theta_i\partial\theta_j}\mu_\theta(X^n)\right]_{\theta=\hat{\theta}}. \tag{50}$$

The second term in the Taylor's expansion is zero as the ML is the maximizer of the likelihood function. For probability densities from the exponential family, the Fisher information matrix is proportional to the inverse of covariance matrix, i.e., $\mathbf{E}_\theta\left[(\theta-\hat{\theta}(y^m))^{\mathrm{T}}(\theta-\hat{\theta}(y^m))\right] = \frac{c_1}{m}\mathcal{I}^{-1}$. Therefore, the third term in the Taylor's expansion would reduce to $c_1\frac{n}{m}$.

The error term follows from the fact that the estimate $\hat{\theta}(y^m)$ is evaluated with a length m sequence and hence the distance $||\theta-\hat{\theta}(y^m)||^2$ is bounded by $\frac{1}{m}$. This concludes the proof. ■

Lemma 7.4.4

$$\mathbf{E}_{X^n}\left[\log\frac{P_{\hat{\theta}(y^m)}(X^n)}{P_{\hat{\phi}(y^m)}(X^n)}\right] = c_2\frac{n}{m} + O\left(\frac{1}{\sqrt{m}}\right),$$

where $c_2 > 0$ is a constant.

Proof The proof of this lemma again requires a second order Taylor's expansion and can be similarly completed with the methods in the proof of lemma 7.4.3. ■

Lemma 7.4.5

$$\begin{aligned} -\log\omega(\hat{\phi}(y^m)) &= -\log w(\hat{\theta}(y^m)) + \log\sqrt{|\mathcal{I}(\hat{\theta}(y^m))|} \\ &\quad +\frac{d}{2}\log m + O(1). \end{aligned}$$

Proof This lemma is obtained as a consequence of Lemma 10.1 in [38] which determines the total number of points in the discretized space. ■

As a consequence of Lemma 7.4.5, we observe that the minimizing prior $w(\theta)$ that results in the minimum description length of the parameter $\hat{\theta}(y^m)$ is the so-called Jefferey's prior:

$$w_{\mathrm{J}}(\theta) = \frac{|\mathcal{I}(\theta)|^{\frac{1}{2}}}{\int_{\theta\in\Theta} |\mathcal{I}(\theta)|^{\frac{1}{2}}\, \mathrm{d}\theta}.$$

Putting it all together, we arrive at the proof of Theorem 7.4.1.

Proof of Theorem 7.4.1 From lemma 7.4.5, the minimum expected communication cost from the helper

$$\mathcal{L}_M = \frac{d}{2}\log m + \int_{\theta\in\Theta} |\mathcal{I}(\theta)|^{\frac{1}{2}}\, \mathrm{d}\theta + o(1).$$

From the lemmas 7.4.3 and 7.4.4, we have

$$\mathcal{L}_S = H_n(\theta^\star) + c\frac{n}{m} + O\left(\frac{1}{\sqrt{m}}\right),$$

where c is a constant. The proof is completed by noting that $\frac{n}{m} = o(1)$ and $\frac{1}{\sqrt{m}} = o(1)$.
■

7.5 Impact of Channel Loss

Wireless networks are prone to errors. Packets are lost due to errors, the wireless channel, limited buffers and congestion. The effect of the packet loss on redundancy elimination in wireless networks is first studied in [52], where the authors show that high loss rate can be detrimental to the redundancy elimination. However, the data gathered from gateways in North American and European wireless service providers show that the average loss rate in the downlink of UMTS providers is around 3% which is well below the loss rate that causes harm to redundancy elimination. We should also point that the average loss rate on the uplink can be as high as 14%, which if not corrected can render redundancy elimination effectively useless. In [52], authors introduced loss recovery schemes which eliminate the adverse effect of loss on redundancy elimiation when the loss rate is high. In particular, the "informed marking" scheme, where each receiver signals the sender whenever it cannot decode a packet due to a missing packet from its cache memory. The receiver sends a control packet of the missing packet and the sender blacklists the corresponding packet in its own cache; in future encoding, any blacklisted packet will be ignored.

Since the focus of this chapter is the compression of the downlink traffic and the average loss rate in the downlink is minimal, the coding techniques discussed in previous sections can be applied in real-world scenarios with no modification. Further, effective schemes such as the informed marking, introduced above, with very low overhead can be employed to counter the loss.

In high loss rate scenarios, more complicated compression schemes should be developed where one might require memory-assisted compression under mismatched side information. The theoretical investigation of the memory-assisted compression with mismatched side information is presented in Chapter 8.

7.6 Simulation

7.6.1 Simulaton Setup

To evaluate the performance of the proposed memory-assisted compression via helpers, we used NS-2 simulator [58]. We employed a flat grid topography with a wireless base-station (S) at the origin. Further, multiple memory-enabled helpers (M) are deployed within the coverage of S. The helpers are uniformly distributed in the coverage of S, which is assumed to be a circle of radius 250m. The communication range of the helpers is 20m and they are placed such that they are outside of the communication range of each other. All the mobile clients are within the communication range of S, but only a subset is covered by helpers at any given time.

We simulate both Constant Bit Rate (CBR) traffic generator over User Datagram

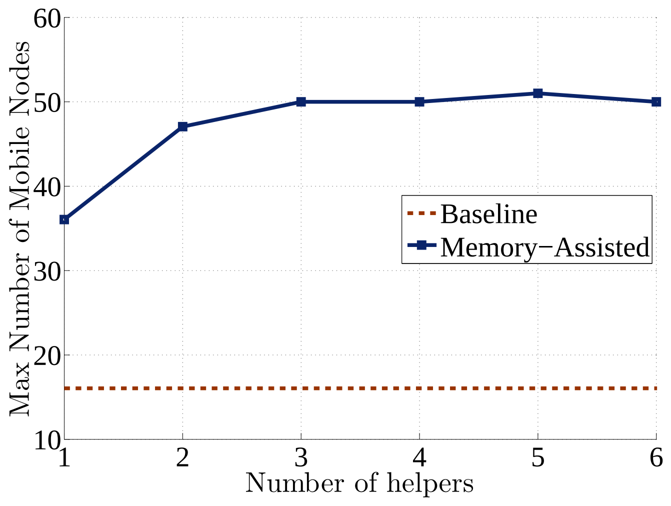


Figure 30: Maximum number of mobile nodes supported by S vs. the number of helpers in the network. The packet drop rate threshold is fixed at 10% and the traffic generator is CBR over UDP, as in Table 5.

Protocol (UDP) and File Transfer Protocol (FTP) which is running over Transport Control Protocol (TCP). We considered the case where S shares a common memory with each of the helpers and that memory is used for compression of packets sent to mobile nodes within the coverage of the corresponding helper. Further, each mobile client (if covered by a helper) only helped by a unique helper node. Obviously, if a node is not in the range of any helper, it receives its packets directly from S (via compression without memory). For the baseline simulation scenario, we consider the case where no helper is deployed and all the communication is conducted by S. Hence, packets are compressed individually without using any memory, i.e., end-to-end compression.

For FTP simulations, we consider files of size 20kbits for which a memory packet of size 2kbits is sent from helper to the client. The details of simulation parameters are given in Table 5.

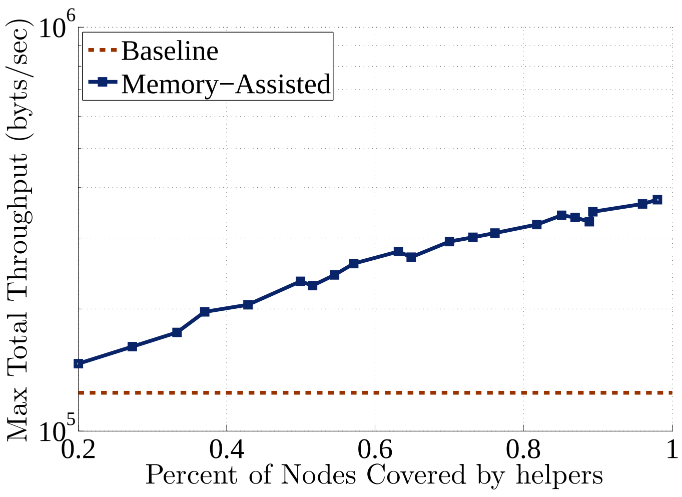
Table 5: Simulation parameters and values

Parameter	**Value**
Number of helpers (M)	$0 - 10$
Comm. Radius of S	250m
Comm. Radius of helper	20m
CBR over UDP rate	64 kbps
UDP baseline packet size	8000 bits
Packet Drop Rate Threshold	10%
FTP file size	20kbit
TCP window size	128

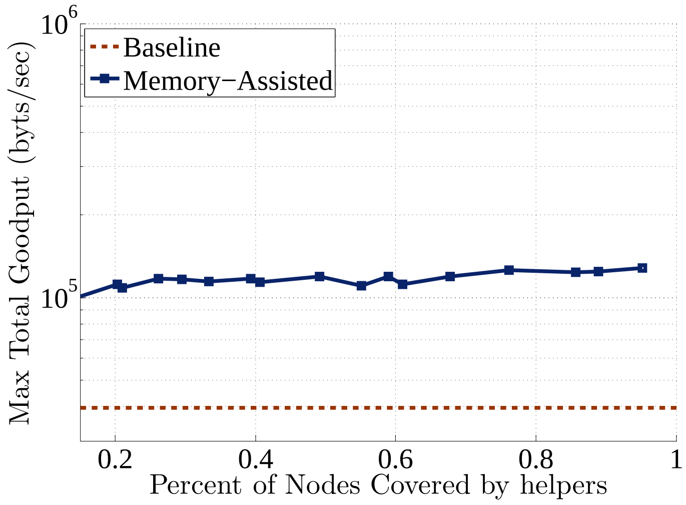
7.6.2 Simulation Results

To examine the effectiveness of the memory-assisted compression, with respect to the baseline scheme, we have considered three performance quantities and evaluated them for both UDP and TCP scenarios. The first quantity is the maximum number of nodes that can be supported, for the traffic described in Sec. 7.6.1. To obtain the maximum number of nodes in Figure 30, we have increased the number of mobile nodes in the environment until the packet drop rate exceeds a 10% threshold. We observe that using memory-assisted compression the maximum number of nodes increases from 15 to almost 50, as shown in Figure 30. Since the bottleneck of the network is the output bandwidth of S, we observe from Figure 30 that adding helpers beyond a certain number does not increase the maximum number of client nodes supported.

In Figure 31, we have depicted the maximum total throughput/goodput versus the fraction of the nodes covered by helpers. For both of the plots in Figure 31, the number of nodes is chosen similar to the setup for Figure 30, that is, the nodes are added to the network (while keeping the helper's coverage constant) until the packet drop rate reaches 10%. The total throughput for UDP traffic and the goodput for TCP trrafic is then measured as the sum over all the clients in the network. As expected, as helpers cover more mobile nodes in the network, higher total throughput is achieved. Since the traffic generation for UDP and TCP scenarios is different, we

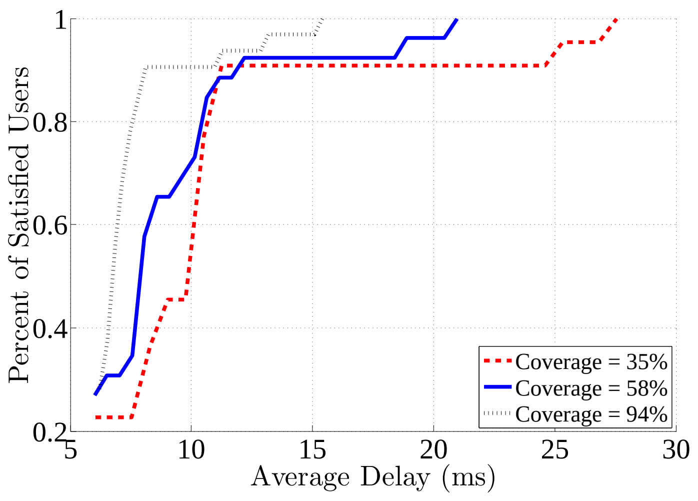


(a)

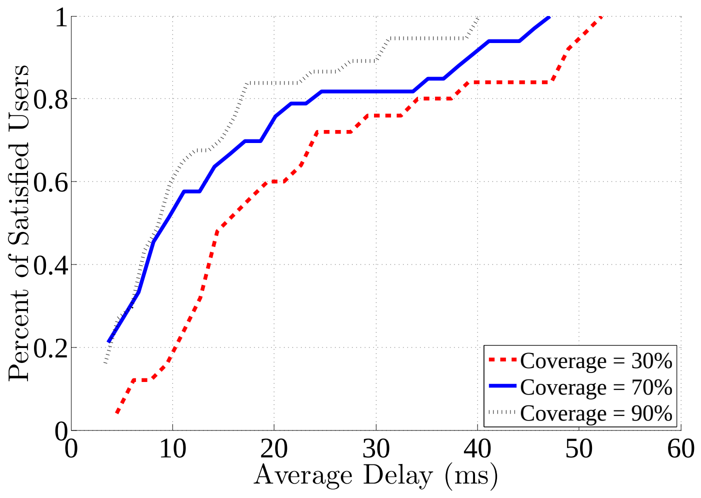


(b)

Figure 31: Maximum total throughput/goodput in the network vs. the fraction of mobile nodes covered by helpers for (a) UDP and (b) TCP.



(a)



(b)

Figure 32: Fraction of satisfied users in the network vs. maximum allowed average delay of packets for (a) UDP and (b) TCP traffic for different helper coverage percentages.

observe different amount of increase in the total throughput/goodput but the trend is increasing for both scenarios as we increase the density of the helpers.

The third quantity of interest is the Quality of Service (QoS). To demonstrate the benefit of memory-assisted compression on QoS, we have considered a simulation scenario with fixed number of clients and measured the average delay of packets for each client. Te number of helper nodes is changed to obtain the plots for different helper coverage ratios. The number of clients per helper is fixed and we have added more helpers to serve more number of clients. Figure 32 depicts the fraction of satisfied clients for a given maximum allowable average delay. As we see, users experience less amount of delay as the fraction of nodes covered by helpers increase.

CHAPTER VIII

MEMORY-ASSISTED COMPRESSION WITH MISMATCHED SIDE INFORMATION

8.1 Introduction

In Chapter 8 the problem of lossless universal compression of finite length sequences with mismatched side information at encoder and decoder is considered. There are two scenarios that motivate the study of memory-assisted compression with mismatched side information at encoder and decoder. First, packet loss in wireless networks can result in mismatched side information; the loss recovery mechanisms in wireless networks do not guarantee perfect delivery for overhearing nodes. Therefore, the overhearing nodes in the networks might not receive all the packets sent by the gateway and when memory-assisted compression is employed, the gateway and client do not have the same side-information. Secondly, in wired networks, the routers do not observe all the packets sent by the source as the routing algorithm may requires packets fro the same source take different paths in the network. In Chapter 8, we take the first steps of characterizing the performance of memory-assisted compression with mismatched side information.

Let x^n be a sequence of short to moderate length n generated by a parametric source, with unknown parameters. Further, assume that the encoder at node S and decoder at node D both have access to some previously generated sequences of the source as a side information. However, this side information at the decoder , denoted by z^m, differs from the side information at the encoder, denoted by y^m, by a number of erasures. Specifically, we assume that D has acquired a number of sequences of total length m from the source, however, some of the symbols in y^m are erased

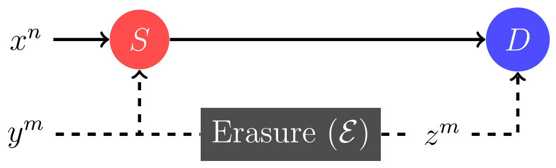


Figure 33: The model abstraction of compression problem with mismatched side information.

before arriving at D, as shown in Figure 33. As such, we aim at answering the following questions: what is the fundamental limit of compression for sending x^n to D given that D has some side information about the parametric source generating x^n? Furthermore, how does the fundamental limit varies with n, m as well as the fraction of erased symbols?

As discussed in Chapter 7, packet-level redundancy elimination in wireless networks can be performed by exploiting the overhearing opportunity in the wireless environment; clients can overhear previously sent data and use the memorized packets (i.e., overheard data) to form a model of the source used in memory-assisted compression. However, the error-prone wireless environment makes it difficult to guarantee that the sender node and the mobile node, which has overheard the previous communications of the source with other mobile clients, have the same model of the information source. This is because the error recovery mechanism is implemented between the source and the client for which the packet is intended to, not the other nodes that overhear the communication. This mismatch between the model at the encoder and the decoder makes the memory-assisted compression challenging. We view the overhearing in the network as an erasure channel, as in Figure 33. In other words, a fraction of symbols from the entire previous sent packets to other clients (i.e. y^m) are erased due to overhearing at D, resulting z^m.

8.1.1 Contributions

We provide theoretical results on the performance of memory-assisted compression when the source model at the encoder and the decoder does not match. In Section 8.3, we explore the impact of erasure on memory-assisted compression in wireless networks and investigate the redundancy of memory-assisted compression in this scenario. The average minimax redundancy is considered as the measure of performance for the universal scheme employed for compression. The idea is to devise a scheme that utilizes the side information to reduce the parameter space in such a way that both the encoder and the decoder arrive at the new parameter space. The reduced parameter space will result in better compression performance. The information-theoretic formulation for the problem in the context of universal compression is presented and bounds on the fundamental limit of the compression performance in this setup is derived. It is shown that the side information, provided that m is sufficiently large, reduces the leading term of redundancy (which is $\frac{1}{2}\log n$) to $O(1)$, if the fraction of erased symbols is sufficiently small. Furthermore, in Section 8.4, a sequential code design is presented for memory-assisted compression with mismatched side information.

8.2 Background and Problem Setup

Consider a node S and two clients C and D in a wireless network where both C and D may receive packets from S, though not perfectly. In a real-world scenario, the node S is a wireless gateway (or tower) that is connected to Internet or the network backbone and transmits the packets to clients in unicast sessions. In the abstraction of the problem, node S may be viewed as a parametric source that sends independent sequences of length n to clients. However, the source parameter is unknown to S and clients. Now, assume that several sequences have already been destined to C via unicast from S, but due to the broadcast nature of the wireless environment, D also overheard some of these sequences. Let y^m be a sequence of length m, which is the concatenation of all previously sent sequences to the other clients by S. Let z^m be the resulting sequence from overhearing y^m by the client D in Figure 33. It is important to note that due to channel erasures, z^m is the same as y^m in Figure 33 except that some of the symbols in z^m are marked as erased. We assume that both S and D know the length of side information m and the number of erasures r, hence, $\mathcal{E} = \frac{r}{m}$ is known. As such, node D has a common side information with S that can be utilized by S toward compression of any new sequence x^n destined to D, as in memory-assisted compression in [13, 15, 14]. However, the fundamental difference of this setup with [13, 15, 14] is that y^m and z^m do not match.

For simplicity of analysis, it is assumed that the source S is a parametric source, over alphabet $\mathcal{A}$, with a d-dimensional parametric vector θ which takes values in $\Theta \subset \Re^d$. One may extend this model to a more realistic setup for real-world sources by considering a mixture model as studied in [15]. Let $x^n = (x_1, \dots, x_n)$ be a sequence generated by the source with probability $\mu_\theta(x^n)$. We wish to study the fundamental limit of compression of x^n, given that S and D have access to some memorized sequences y^m and z^m, respectively. Further, it is of interest to characterize as to how this limit vary as n and m as well as the erasure fraction change. In the absence of memory-assisted compression, i.e., compression without using the common memory, x^n is universally coded by $c_n : \mathcal{A}^n \to \{0,1\}^*$ with the length function denoted by $l(x^n)$. To proceed, we state the two notions of lossless compression as defined below.

Definition The code $\hat{c}_n(\cdot) : \mathcal{A}^n \to \{0,1\}^*$ is called almost lossless if there exists a reverse mapping $\hat{d}_n(\cdot) : \{0,1\}^* \to \mathcal{A}^n$ such that

$$\lim_{n\to\infty} \mathbf{E}\{\mathbf{1}_{err}(X^n)\} = 0,$$

where $\mathbf{1}_{err}(x^n)$ denotes the indicator function of error, i.e,

$$\mathbf{1}_{err}(x^n) = \begin{cases} 1 & \hat{d}_n(\hat{c}_n(x^n)) \neq x^n, \\ 0 & \text{otherwise.} \end{cases}$$

Definition The code $c_n(\cdot) : \mathcal{A}^n \to \{0,1\}^*$ is called strictly lossless if there exists a reverse mapping $d_n(\cdot) : \{0,1\}^* \to \mathcal{A}^n$ such that

$$\forall x^n \in \mathcal{A}^n : \quad d_n(c_n(x^n)) = x^n.$$

All of the practical data compression schemes are examples of strictly lossless codes, namely, the arithmetic coding, Huffman coding, Lempel-Ziv algorithm, and the Context Tree Weighting [84]. In Chapter 8, we only consider the universal coding c_n that are strictly lossless.

The performance of the compression employed is measured in terms of the overhead beyond the entropy, which is called the average code redundancy as

$$R(c_n, \theta) = \mathbf{E}[l(X^n)] - H_n(\theta),$$

where $H_n(\theta)$ is the entropy of the parametric source induced by μ_θ on x^n.

If the parameter vector $\theta \in \Theta$ was known, the ideal code length of a sequence x^n, obtained from the Shannon code (ignoring the integer code length requirement), would be $\log \frac{1}{\mu_\theta(x^n)}$. Without the knowledge of θ, one has to encode the sequence with a penalty term that is characterized by the code redundancy. The average minimax redundancy, defined as

$$\bar{R}(n, \Theta) = \min_{c_n} \max_{\theta \in \Theta} R(c_n, \theta),$$

is a performance measure for universal lossless coding schemes. It is shown in [10, 66] that

$$\bar{R}(n, \Theta) = \frac{d}{2} \log \left(\frac{n}{2\pi e} \right) + \log \int_{\theta \in \Theta} |\mathcal{I}(\theta)|^{\frac{1}{2}} \, \mathrm{d}\theta + O\left(\frac{1}{n} \right), \tag{51}$$

where $|\mathcal{I}(\theta)|$ is the determinant of the Fisher information matrix evaluated at θ.

A related result to (51) that is used in this work is the following theorem [38, 30]:

Theorem 8.2.1 *Let $\Theta_0 \subset \Theta$. Denote by $\bar{R}(w,\theta)$ the expected redundancy of a compression scheme with the prior $w(\theta)$ on Θ_0. Then, we have*

$$\bar{R}(w,\theta) = \frac{d}{2}\log\left(\frac{n}{2\pi e}\right) - \log w(\theta) + \log |\mathcal{I}(\theta)|^{\frac{1}{2}} + o(1), \tag{52}$$

where the convergence is uniform in $\theta \in \Theta_0$.

This theorem suggests that Jeffreys' prior, defined as

$$w_{\mathrm{J}}(\theta) = \frac{|\mathcal{I}(\theta)|^{\frac{1}{2}}}{\int_{\theta\in\Theta_0} |\mathcal{I}(\theta)|^{\frac{1}{2}}\, \mathrm{d}\theta}, \tag{53}$$

is maximin optimal. Note that Jeffreys' prior is also minimax optimal [56].

Finally, another important relationship that we use in this chapter is the following result by Gallager which shows that if μ_θ is a measurable function of θ, then

$$\bar{R}(n,\Theta) = \sup_{w(\theta)} I(X^n;\theta), \tag{54}$$

where $I(X^n;\theta)$ is the mutual information between X^n and θ, and $w(\theta)$ is the prior distribution on θ.

8.3 Main Results

We note that for finite length sequences, there is a large gap between the code length of the best universal code and the source entropy, i.e., the average code redundancy is considerable [12]. However, the side information z^m obtained by overhearing, provides information regarding the parameter θ and hence can be used to reduce the code redundancy considerably. We are after the fundamental improvement as well as practical code designs that exploits the side information to get close to optimal universal code lengths and is easy to implement.

Let $\bar{R}(n,\mathcal{E})$ be the average minimax redundancy of the compression employed in

Figure 33, where $\mathcal{E} = \frac{r}{m}$ is the fraction of erased symbols in y^m. First, we obtain a lower limit on the average minimax redundancy for the case when the side information at both ends is exactly the same. In other words, the encoder exactly knows the location of erased symbols, i.e., the encoder knows z^m. In this case, we have

Proposition 8.3.1 *If the location of r erased symbols in the side information is known at the encoder, i.e., node S in Figure 33, then we have*

$$\bar{R}(n, \mathcal{E}) \geq \frac{d}{2} \log \left(1 + \frac{n}{m-r}\right) + O\left(\frac{1}{n}\right). \tag{55}$$

Proof Let $\bar{R}(n|\lambda)$ be the average minimax redundancy of a universal scheme compressing a sequence of length n given a side information v^λ, and $\lambda > n$ known to both encoder and decoder. According to [14] and using (54), the minimax redundancy of a memory-assisted compression scheme with a side information of size λ can be obtained as

$$\begin{aligned}
\bar{R}(n|\lambda) &= \max_{p(\theta)} I(X^n; \theta | V^\lambda) \\
&= \max_{p(\theta)} \left[I(X^n, V^\lambda; \theta) - I(V^\lambda; \theta) \right] \\
&= \bar{R}_{n+\lambda} - \bar{R}_\lambda.
\end{aligned} \tag{56}$$

From (51), we have
$$\bar{R}(n|\lambda) = \frac{d}{2} \log \left(1 + \frac{n}{\lambda}\right) + O\left(\frac{1}{n}\right).$$

The result in (55) follows from (56) and the fact that the destination has only access to a memory of size $\lambda = m - r$ and no strictly lossless compression scheme can benefit from a side information longer than the one available at the destination. ■

This is a trivial upper bound, which is tight when the erasure $\mathcal{E} \to 1$, i.e., there is no memory shared between S and the client. This bound is obtained by ignoring the available memory at the encoder and the decoder. Then, $\bar{R}(n,\mathcal{E})$ is bounded from above by

$$\bar{R}(n,\mathcal{E}) \leq \bar{R}(n). \tag{57}$$

The upper bound introduced in (57) is not tight for all $\mathcal{E}$. In the following, we provide a constructive approach which leads to a non-trivial upper bound on the average minimax redundancy when the source is memoryless. The bound is given for binary case for the simplicity of the presentation but the extension to non-binary alphabet is straightforward.

Theorem 8.3.2 *The average minimax redundancy of a universal compression scheme for the class of binary memoryless sources with a side information obtained through a channel with a fraction $\mathcal{E}$ of symbols erased is bounded by*

$$\begin{aligned}\bar{R}(n,\mathcal{E}) \leq \sum_{i=1}^{\min(\lceil \frac{1}{\mathcal{E}} \rceil, 1)} & \frac{2}{\pi}\left(\arcsin\sqrt{i\mathcal{E}} - \arcsin\sqrt{(i-1)\mathcal{E}}\right) \\ & \left(\frac{1}{2}\log\frac{2n}{\pi e} + \log \mathcal{C}_{(i-1)\mathcal{E}}^{i\mathcal{E}} + o(1)\right),\end{aligned} \tag{58}$$

where

$$\mathcal{C}_{\alpha_1}^{\alpha_2} = \int_{\alpha_1}^{\alpha_2} \frac{1}{\sqrt{x(1-x)}}\, dx.$$

Proof Consider a memoryless source with parameter space $\Theta = (0,1)$ and alphabet $\mathcal{A} = \{a,b\}$. The Jeffreys' prior for this source, defined in (53), is $w(\theta) = \frac{1}{\pi\sqrt{x(1-x)}}$. If we use this prior for coding a sequence x^n, the resulting redundancy would be

$$\bar{R}(n,\Theta) = \frac{1}{2}\log(\frac{n}{2\pi e}) + \log(\pi) + o(1).$$

However, the side information will induce another prior on the parameter space that reduces the redundancy of the Jeffreys' prior. Consider a sequence y^m at S with $m_S^{(a)}$ number of a's. Likewise, let $m_D^{(a)}$ be the number of a's in z^m. Let $\hat{\theta}_S$ denote the ML estimate of θ at S and $\hat{\theta}_D$ be the ML estimate at D. We have

$$\begin{aligned} \hat{\theta}_S &= \frac{m_S^{(a)}}{m} \\ \frac{m_S^{(a)} - r}{m} \leq \hat{\theta}_D &= \frac{m_D^{(a)}}{m} \leq \frac{m_S^{(a)}}{m}. \end{aligned} \tag{59}$$

A strictly lossless compression scheme requires both the encoder and the decoder use the same parameter estimate or prior. To overcome the mismatch in (59) between $\hat{\theta}_S$ and $\hat{\theta}_D$, we consider the following scheme: both the encoder and the decoder divide the interval $(0, 1)$ into sub-intervals of size $\frac{r}{m}$. Since $\hat{\theta}_D \leq \hat{\theta}_S$ and $|\hat{\theta}_D - \hat{\theta}_S| < \mathcal{E}$, the estimated parameter at the encoder and the decoder are either in the same sub-interval or in two adjacent sub-intervals. This discrepancy can be resolved with one extra bit sent by the encoder.

Let $\Theta_i = ((i-1)\mathcal{E}, i\mathcal{E})$ be the i-th sub-interval. Since, $w_J(\theta)$ is Jeffreys' prior,

$$\begin{aligned} \mathbf{P}[\theta \in \Theta_i] &= \int_{\theta \in \Theta_i} w_J(\theta)\, \mathrm{d}\theta \\ &= \frac{2}{\pi}\left(\arcsin\sqrt{i\mathcal{E}} - \arcsin\sqrt{(i-1)\mathcal{E}}\right). \end{aligned}$$

Further, for binary memoryless sources, $\mathcal{I}^{-1}(\theta) = \theta(1-\theta)$. Hence, according to Theorem 8.2.1, the redundancy of a compression scheme, with the side information that the source parameter is chosen from Θ_i, can be obtained as

$$\begin{aligned} \bar{R}(n, \Theta_i) &= \frac{1}{2}\log\left(\frac{n}{2\pi e}\right) + \log\int_{\theta \in \Theta_i} |\mathcal{I}(\theta)|^{\frac{1}{2}} + o(1) \\ &= \frac{1}{2}\log\left(\frac{n}{2\pi e}\right) + \log \mathcal{C}_{(i-1)\mathcal{E}}^{i\mathcal{E}} + o(1). \end{aligned}$$

Thus, the result in (58) follows. ■

It is easy to see that the bound in (58) reduces to (51) for $\mathcal{E} \to 1$. This is because $\mathcal{C}_0^1$ reaches its maximum of π and the total number of sub-intervals within $(0, 1)$ is one and hence no extra information is needed from the source.

For the special case of $\mathcal{E} = O(\frac{1}{\sqrt{n}})$, we can provide a stronger result that exactly characterizes $\bar{R}(n, \mathcal{E})$.

Proposition 8.3.3 *If $\mathcal{E} = O(\frac{1}{\sqrt{n}})$, we have*

$$\bar{R}(n, \mathcal{E}) = O(1). \tag{60}$$

Proof Since $\mathcal{E} = O(\frac{1}{\sqrt{n}})$, the size of sub-interval Θ_i is also $O(\frac{1}{\sqrt{n}})$. Let $\theta^\star \in \Theta_i$, then,

$$\begin{aligned} \bar{R}(n, \Theta_i) &= \mathbf{E}[\log \mu_\theta(X^n) - \log \mu_{\theta^\star}(X^n)] \\ &= nD(\mu_\theta || \mu_{\theta^\star}) \\ &\overset{(i)}{=} \frac{n}{2}(\theta - \theta^\star)^2 \mathcal{I}(\theta) + o(1) \\ &\overset{(ii)}{=} O(1), \end{aligned} \tag{61}$$

where $D(.||.)$ is the KL divergence. In (61), equality (i) follows from the second order approximation of KL divergence and (ii) follows from the fact that $(\theta - \theta^\star)^2 < \frac{1}{n}$. Finally, (60) is derived from (61). ■

8.4 Discussion on Practical Code Design

Construction of a memory-assisted compression scheme with mismatched side information for a memoryless source easily follows from the proof of Theorem 8.3.2. As the proof suggests, we should first construct a code that would compress a sequence with the side information that the parameter is from a sub-interval Θ_i. Let $x^t = x_1 x_2 \dots x_t$ be a sequence with binary symbols. Clearly,

$$\mathbf{P}\left[x^t | \theta\right] = \theta^{n_a(t)} \theta^{t - n_a(t)},$$

where $n_a(t)$ is the number of symbols a in x^t. The prior probability on the sub interval Θ_i is the normalized Jefferys' distribution, i.e.,

$$w_J(\theta) = \frac{|\mathcal{I}(\theta)|^{\frac{1}{2}}}{\mathcal{C}_{(i-1)\mathcal{E}}^{i\mathcal{E}}}.$$

Therefore, the probability of the sequence x^t is equal to

$$\mathbf{P}\left[x_1^t\right] = \int_{(i-1)\mathcal{E}}^{i\mathcal{E}} \frac{1}{\mathcal{C}_{(i-1)\mathcal{E}}^{i\mathcal{E}}\sqrt{\theta(1-\theta)}}\theta^{n_a(t)}\theta^{t-n_a(t)}\,\mathrm{d}\theta. \tag{62}$$

Now, the sequence x^n along with its probability can be passed to an arithmetic encoder. However, from a practical point of view, a better compression scheme can be used which evaluates the probability in (62) sequentially. As such, the sequential probability estimates of x^{t+1} can be evaluated as follows [76]:

$$\begin{aligned}\mathbf{P}\left[x_1^{t+1}\right] \quad &= \quad \mathbf{P}\left[x_1^t\right]\frac{n_{x_{t+1}}(t)+\frac{1}{2}}{t+1}\\ &\quad +\beta\times\frac{{\alpha_1}^{n_a(t)+\frac{1}{2}}(1-\alpha_1)^{n_b(t)+\frac{1}{2}}}{\mathcal{C}_{\alpha_1}^{\alpha_2}(1+t)}\\ &\quad -\beta\times\frac{{\alpha_2}^{n_a(t)+\frac{1}{2}}(1-\alpha_2)^{n_b(t)+\frac{1}{2}}}{\mathcal{C}_{\alpha_1}^{\alpha_2}(1+t)},\end{aligned}$$

where $\alpha_1 = (i-1)\mathcal{E}$, $\alpha_2 = i\mathcal{E}$, and

$$\beta = \begin{cases} 1 & x_{t+1} = a, \\ -1 & x_{t+1} = b. \end{cases}$$

The results in Figure 34 show the performance of the proposed sequential compression scheme for different sequence lengths, fraction of erased symbols, and side information of length $m = 10^6$. The quantity g in Figure 34 is defined as the gain of memory-assisted compression (with side information) over compression with no side information, i.e., $g \triangleq \frac{\mathbf{E}l(X^n)}{\mathbf{E}l(X^n|Z^m)}$. We observe that for simple binary sequences of

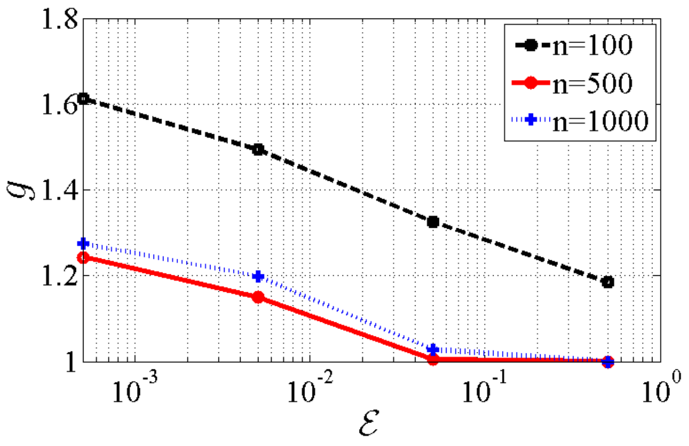


Figure 34: Gain of memory-assisted compression over the end-to-end compression. The memory size is $m = 10^6$.

length 100, a compression gain of 1.6 on top of the end-to-end compression of x^n is achieved, on the average. This gain is expected to be higher if we consider non-binary sources, as observed in [15].

CHAPTER IX

MEMORY-ASSISTED COMPRESSION WORKING IN TANDEM WITH DE-DUPLICATION

9.1 Introduction

In Chapter 9, the use of preprocessing for data compression is investigated. Preprocessing is often used to find and remove the long repeated sequences that appear in the data to be compressed [16]. As such, in this work we refer to preprocessing as *de-duplication.* Network traffic often contains large repeated blocks. For example, traffic from users that access the same web server contains a large number of repeated blocks and de-duplication is an effective tool to find these blocks. The benefits of de-duplication are complementary to those offered by the main data compression algorithms. De-duplication provides a fast and efficient way of removing repetitions that are far apart. Such repetitions remain undetected in the absence of the de-duplication because of the physical memory constraints of the main compression engine. Furthermore, removing the input size has a positive impact on the speed of the compression engine.

In short, preprocessing in the form of de-duplication is an effective technique to boost the speed and overall compression performance of compression algorithms while reducing the memory footprint. Furthermore, since the preprocessing is designed independent of the compression engine, there the compression algorithm remains unchanged. In Section 9.2.1 and Section 9.2.2, the details of the de-duplication algorithms are discussed. It is demonstrated in Section 9.2.3 that the de-duplication algorithm has to take into account the content of the packets because a simple fixed-block de-duplication proves to be ineffective. Hence, it is demonstrated that a value-based de-duplication is the method of choice.

In Section 9.3, the performance of compression algorithms after de-duplication is studied and visual graphs are provided to demonstrate the impact of the content type on the outcome of de-duplication and compression algorithms. We further, employ the joint memory-assisted compression and clustering after the de-duplication and provide compression results that enhance the benchmark on the state-of-the-art redundancy elimination techniques.

9.1.1 Contributions

In Chapter 2 the benefits of memory-assisted compression was established and in Chapter 3 the benefits of joint clustering and memory-assisted compression was demonstrated. In Chapter 9, the benefits of preprocessing for improving the performance of memory-assisted compression are investigated. In particular, we study the use of de-duplication for removing the repetition in the data. Experiment results presented demonstrate the effectiveness of de-duplication for removing the dependencies that are to detected by data compression algorithms. The experiments are performed on real-world data traces collected from thirty mobile users over a month long period.

9.2 Preprocessing for Data Compression

Preprocessing is often used to find and remove the long repeated sequences that appear in the data to be compressed. Removing the repeated sequences before feeding the input sequence to the main compression engine provides three major benefits for data compression:

1. **Speed**: The use of fast preprocessing for speeding up the high-performance but low-speed statistical data compressors is a common practice. The number of operations that a high-performance compression algorithms perform on the

input data scales linearly with the size of input, however, this scaling has a large constant. Therefore, it is suitable to reduce the size of input by removing the repetitions with fast preprocessor and then feed the outcome to the high-performance compressor. For example, fast dictionary-based compressors such as LZP [20] are used before a strong statistical compression algorithm, such as PAQ, for both speed and compression ratio improvement.

2. **Compression performance**: Dictionary-based compression schemes usually employ a sliding window, typically a few hundred kilobytes long. Increasing the size of the window would increase the number of bits required to describe a position in the window and also makes the searching more difficult. Hence, repetitions are discovered if the repeated sequences appear in the same window that is less than a megabytes long. The main drawback of this approach is that the repetitions that are far apart remain undetected. Also, in statistical compression schemes, finding the exact repetition is handled with a table of limited size and new entries replace the old ones in the table. As such, repetitions that are far from each other are missed.

3. **Memory footprint**: For the main compression algorithm to exploit all the repetitions in the data, the memory footprint of the algorithm would scale linearly with the size of the input. This is particularly undesirable when the input size is very large. Preprocessing helps to reduce the memory footprint of the main compression algorithm while maintaining the same level of or even better compression performance.

In short, preprocessing in the form of de-duplication is an effective technique to boost the speed and compression performance of compression algorithms while reducing the memory footprint. In Section 9.2.1 and Section 9.2.2, the details of the de-duplication algorithms are discussed.

9.2.1 Pattern Matching

The core to de-duplication is an efficient algorithm to fingerprint chunks of data and use those fingerprints to find the repetitions. In [43], Karp and Rabin originally presented their pattern matching algorithm for string searching; the algorithm answers whether a pattern sequence $\sigma_1, \ldots, \sigma_l$ exists in a packet of length n. The pattern is viewed as a polynomial of degree l over the finite field of characteristic 2. We have

$$f(\sigma_1, \ldots, \sigma_l) = \sigma_1 x^l + \sigma_2 x^{l-1} + \ldots + \sigma_l \tag{63}$$

The fingerprint of the pattern is defined as the remainder of division of $f(\sigma_1, \ldots, \sigma_l)$ by an irreducible polynomial of degree k over a finite field of characteristic 2. Hence, the fingerprint can be viewed as a polynomial of degree $k-1$. The algorithm computes (63) for each of the substrings of length l of the input packet. The definition of fingerprints in (63) makes the successive computation of fingerprints computationally feasible. For example, in order to compute the fingerprint for the next pattern $\sigma_2, \ldots, \sigma_{l+1}$ we have

$$\begin{aligned} f(\sigma_2, \ldots, \sigma_{l+1}) &= \sigma_2 x^l + \sigma_3 x^{l-1} + \ldots + \sigma_{l+1} \\ &= \left(f(\sigma_1, \ldots, \sigma_l) - \sigma_1 x^l\right) x + \sigma_{l+1}. \end{aligned} \tag{64}$$

As such, a fingerprint can be initialized in $O(l)$ time and updated by sliding one position in $O(1)$ time. The finger printing process facilitates the de-duplication by providing an easy to compute function that can quickly lead to potential duplicates in the packets.

It is impractical to store all fingerprints that are generated by sliding a window of length l over the input packet, since the number of generated finger prints is equal to the length of the input which in turn result in memory blowup. To solve this problem, in the simplest form, the de-duplication algorithms divide the input packets into non-overlapping blocks of length l and store the fingerprint of each block in an easily searchable data structure. That is, the fingerprint of bytes $1, \dots, l$ and $l+1, \dots, 2l$ and so forth, are computed and stored. Therefore, for an input length n, this method stores approximately $\frac{n}{l}$ fingerprints. Using larger values of l can dramatically decrease memory requirements but slightly decreases the de-duplication efficiency. We call this approach "fixed-block fingerprinting".

The above algorithm is implemented and investigated in [16]. In [16], the algorithm uses a default blocks of size 100 bytes and from every block a 32-bit fingerprint is stored. These fingerprints are used to detect repeated strings. The repeated string is represented as "$< start, length >$", where *start* is the initial position and *length* is the size of the sequence. Figure 35 provides a schematic of the de-duplication algorithm. The outcome is then fed into a standard compression algorithm that efficiently represents short (and near) repeated strings. The algorithm of Karp and Rabin has been vastly adopted in various applications, for example [27, 48], to name a few.

9.2.2 Value-base Fingerprinting

One of the main drawbacks of the algorithm presented in Section 9.2.1 is the use of fixed block size, i.e., the input sequence is divided into non-overlapping blocks of fixed size and fingerprints are only stored for those blocks. Consider two packets that are identical except that one has an extra byte inserted at the beginning. The algorithm in Section 9.2.1, initializes at the beginning of each packet, computes and then stores the fingerprints at fixed intervals. As such, none of the stored fingerprints for packets match and as a result no duplicate is found by the algorithm.

To remedy this problem, it is important to store a set of fingerprints that are more effective for finding duplicates in network packets. In [80, 54], it is suggested to store the fingerprints according to their values, i.e., it is desirable to select a

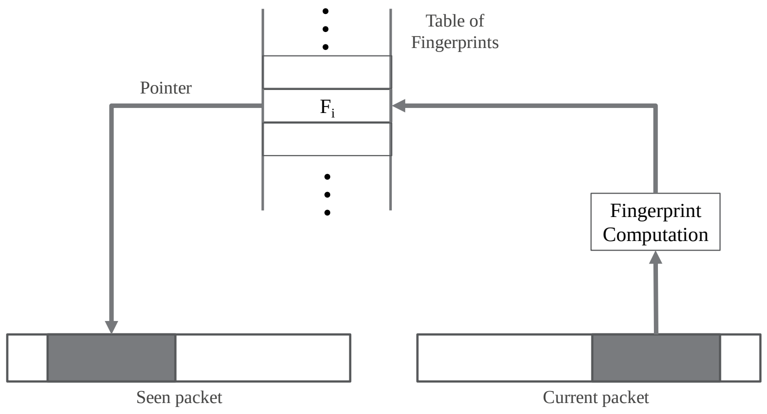


Figure 35: Schematic of the de-duplication algorithm using fingerprinting for pattern matching.

fraction of all the fingerprints generated from the input based on their values and not sensitive to location of the fingerprint. To this end, a prime number n_p is selected and those fingerprints that their remainder modulo n_p evaluate to zero are stored in the table. Therefore, a fraction of $\frac{1}{n_p}$ of fingerprints are uniformly sampled and stored irrespective of the different ways the content is packetized. Selecting a fraction of fingerprints reduces the sensitivity to location and provides a sample of content of the packets. It is shown in [80] that for removing duplicates from network packets of size $\approx$1500 bytes, one can use blocks of size $\approx$100 bytes and sample 2^{-5} of fingerprints to effectively find a large portion of duplicates.

9.2.3 Experimental Results

The objective of the simulations in Section 9.2.3 is to compare the performance of fixed-block and value-based fingerprinting algorithms. The simulation results presented in Chapter 9, including the results in this section are performed on the data set provided in [68]. The data set includes real network traffic collected from 30 different mobile users consisted of smartphone users and laptop users. The laptop users relied only on WiFi connectivity for their network access. The smartphone users relied on both WiFi and 3G connectivity. The data collection spanned a period of 3 months and yielded over 26 Gigabytes of unsecured down link data. Users accessed the Internet as per their normal behavior. More details about the acquisition process can be found in [68].

The summary of the experiments is presented in Table 6. Both de-duplication algorithms are performed on the whole traces in the data set. The fixed-block de-duplication results are based on the evaluation of the algorithm provided in [16] with a default block length of 100 bytes and 32-bit fingerprints. The value-based de-duplication algorithm used is from [68] that uses a large prime number (1048583) for calculating the Rabin fingerprints. However, to reduce the number of bits to represent the repeated strings, the algorithm uses a more efficient Jenkins hash of each repeated sequence. In other words, the Rabin fingerprints are used to find the potential repeated sequences within the packets an for communicating the repeated sequences, an 8-byte Jenkins hash is used. The results of the experiments clearly indicate that fixed block de-duplication is not suitable for removing duplicates from network packets; fixed-block algorithm fails at detecting the duplicates and the overhead of the algorithm results in an output larger than the original input trace. In contrast, value-based fingerprinting is capable of reducing the input size by more than 15%. The value-based algorithm demonstrates superior performance in terms of de-duplication ratio $r = \frac{\text{output size}}{\text{input size}}$. In particular, in some cases, it is capable of reducing the input size by half. This experiment demonstrates the importance of the location sensitivity inherent in the fixed-block algorithm. Hence, in the future experiments of Chapter 9, we only use the value-based de-duplication algorithm.

Table 6: The comparison of de-duplication performance of fixed-block and value-based fingerprinting. The ratio r is defined as $r = \frac{\text{output size}}{\text{input size}}$.

Method	mean(r)	min(r)	std(r)
Fixed-block	1.004	1.000	0.002
Value-based	0.849	0.431	0.145

9.3 Performance Evaluation of Compression Algorithms After Preprocessing

In Section 9.3, we investigate the compression performance of various compression algorithms on the data set introduced in Section 9.2.3 after the preprocessing. The results are presented for each user in the data set. The packet size in each trace varies and is not fixed, however the packets are always less that 1500 bytes. Each trace of each user is processed individually and the compression is performed on the whole trace data. No clustering is performed on the data packets. In Figure 36, the compression performance of a dictionary-based compression algorithm (Gzip ran on Linux with default parameters) and a statistical compression algorithm (lite PAQ ran on Linux with option 4) is demonstrated. The first bar shows the outcome of de-duplication via value-based fingerprinting, denoted by DD in short. The second bar is the compression result of the trace data using Gzip algorithm.

It is important to note the unexpected behavior of dictionary-based compression that in some cases results in an output larger in size than the de-duplication algorithm, an algorithm which is noticeably simpler than Gzip. This behavior can be justified by a closer examination of the trace content. In Figure 37 and Figure 38 the visualization of the user's trace data is depicted for users 1, 10, 11, and 12. For users 10 and 12, the de-duplication outperforms the Gzip and for users 1 and 11, it is the other way round. It is important to observe that the visualization in Figure 37 and Figure 38 demonstrate clear difference in the structure of user data for these two sets of traces.

In Figure 37, presence of green and blue pixels that indicate the presence of long duplicates that are tens of megabytes apart is the main reason that Gzip under performs. This is because these duplicates remain undetected as they do not appear in the search window of the Gzip which is less than a few megabytes long.

On the other hand, for users 1 and 11, it seems that long repeated sequences are scarce, as pixels of black and red color dominate the area in Figure 38. In such cases,

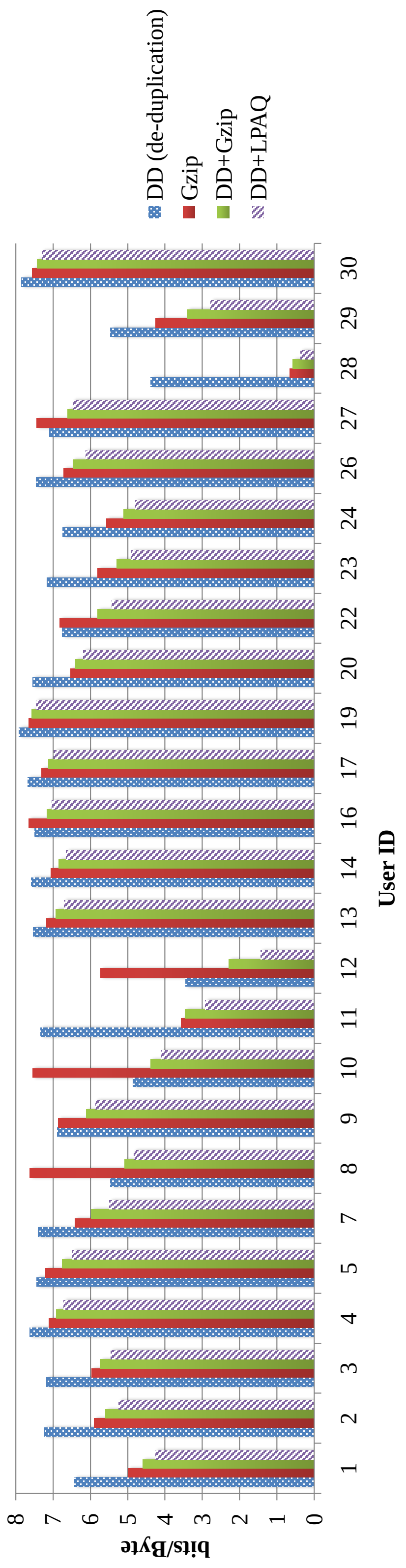
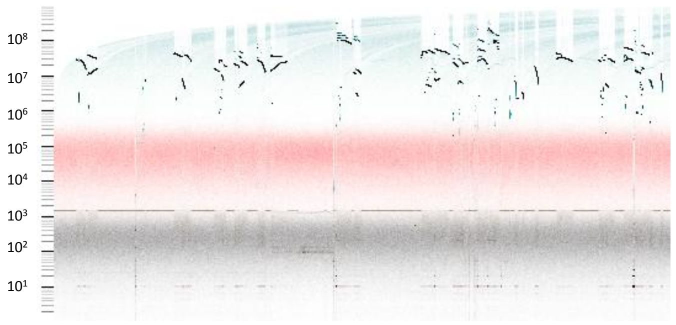
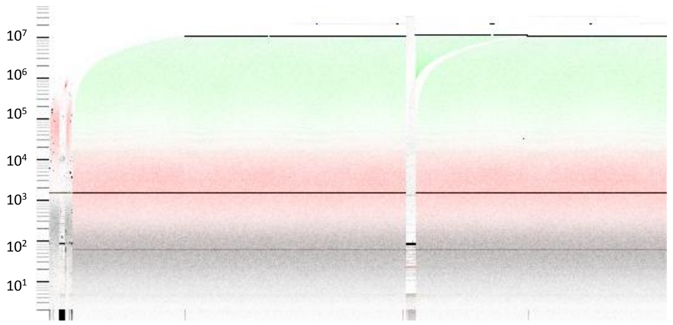


Figure 36: Performance of various compression algorithms on the data set introduced in Section 9.2.3.

de-duplication algorithms do not perform well while Gzip efficiently takes advantage of short term dependencies in the data.



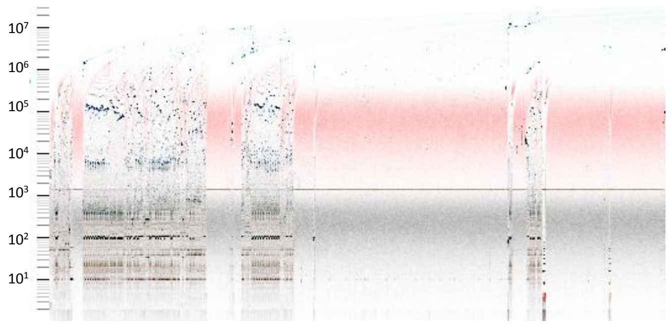
(a) Visualization of trace data for user 10



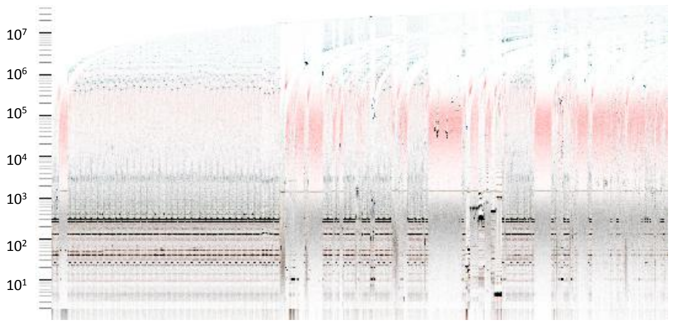
(b) Visualization of trace data for user 12

Figure 37: Visualization of the user data traces, for users with ID 10 and 12. Each pixel represents a duplicate string (consecutive byte sequence). The color of the pixel represents the length of the match: black for 1 byte, red for 2, green for 4 and blue for 8. The horizontal axis represents the position of the second occurrence of the string from the beginning. The vertical axis represents the distance back to the match on a $\log_{10}$ scale.

This result reaffirms the need for using the de-duplication and compression algorithms together to capture both far apart as well as the dependencies that are closer together. In Table 7, the results of the Figure 36 are uniformly averaged over users and the outcome is presented. The best performance is demonstrated by de-duplication used in tandem with lite PAQ statistical compression algorithm whose details are presented in Appendix A.



(a) Visualization of trace data for user 1



(b) Visualization of trace data for user 11

Figure 38: Visualization of the user data traces, for users with ID 1 and 11.

Table 7: Summary of performance of various de-duplication and compression algorithms presented in Figure 36. The results are reported as the ratio $\frac{\text{output size}}{\text{input size}}$.

Value-based de-duplication (DD)	**Gzip**	**DD+Gzip**	**DD+LPAQ**
0.849	0.785	0.698	0.659

9.4 Application of Clustering and Memory-assisted Compression for Compression of Mobile User Data Traces

In order to evaluate the performance of clustering and memory-assisted compression for the training and compression of data traces of mobile users, we need to adapt the statistical compression implementation to compress a large number of test packets by training on the memory just once. We changed the source code of the LPAQ algorithm following the suggestion of [44]. For memory-assisted compression without clustering (UcompM), after the training on the memory is performed, the “fork()”

system call is used to copy the process for every test packet. By this technique, the state of the compression algorithm after training is exactly copied for all the test packets and allows each test packet to be compressed independently. After this, the model is updated for each test packet as new bytes of the test packet are read, however, the processes for different packets remain independent of each other.

Clustering is performed using the technique discussed in Section 3.6. The experiment is performed per user. We have first extracted the features for every packet in the study. The data (of each user) is then divided into two disjoint train and test sets. The packets in the train set are then clustered using the non-parametric clustering method (DPGMM from Scikit Learn library [59] which is an infinite mixture model with the Dirichlet Process as a prior distribution on the number of clusters) with maximum number of clusters set to 50 and all other parameters set to default. The memory of memory-assisted compression is then formed by grouping the packets belonging to the same cluster together. The test packets are then classified (via "predict()" method from DPGMM class) and the cluster index of each test packet is obtained. The compression is then performed by training the statistical compression method for each cluster and then fork the process to compress the test packets deemed to belong to the cluster.

9.4.1 Experiment Results

To characterize the gain of joint clustering and memory-assisted compression, we have considered two scenarios for experiments. In both scenarios, we first remove the far apart duplicates using the value-based de-duplication algorithm discussed in Section 9.2.2. In the first scenario, the training packets are sampled uniformly from the data trace of each user (after de-duplication). Total of 10000 packets are sampled as training data and 1000 packets (disjoint from the training set) are sampled as test packets. The packet sizes are different and the maximum packet size is 1500 bytes.

The size of training data is around 10 MB, on the average.

In the second scenario of the experiment, the first 90% of the data packets in the beginning of each users' trace are chosen as training set and the remaining 10% are test packets. As the data in each user's trace is not stationary, i.e., the statistics of the data changes over time, the results of these two series of experiments would not be the same. However, the trends in these experiments can be used as an indication of the performance of the memory-assisted compression in real scenarios.

The results of the experiments are presented in Table 8. The traditional end-to-end packet compression (Ucomp) provides no improvement and have inflated the packets as the average compression rate is more than 8 bits per byte. The memory-assisted compression enhances the compression performance drastically as shown in the third column of Table 8. Furthermore, clustering provides an additional improvement on the compression performance which is in accordance to the results of the Section 3.7.1.

Table 8: Results of the joint memory-assisted compression and (non-parametric) clustering after de-duplication for mobile users. The results are reported in bits/byte.

Experiment Series	Ucomp	UcompM	UcompM+Clustering
Scenario 1	8.603	5.940	5.732
Scenario 2	8.020	5.902	5.674

9.5 Impact of Memory Footprint on Compression Performance

In Section 9.2, the physical memory requirement of a compression algorithms was mentioned as an important factor to evaluate the compression algorithm. In Section 9.5 the trade-off between the amount of physical memory used by a statistical compression algorithm and its compression performance is investigated. In particular, we study two statistical compression algorithms, namely, LPAQ and CTW. In some wireless applications, it is desirable that clients pass the state of the compressor they locally maintain to the gateway such that the gateway and the client can synchronize their compressors and improve compression performance. The study of trade-off between the size of the physical memory and the compression performance is motivated by such applications.

The CTW algorithm maintains a tree model for the data and hence the number of nodes in the tree determines the amount of memory used by the algorithm. A tree structure in computer memory can be stored in various ways:

- Allocating an array in which the data of all possible nodes in a fully extended tree (to a certain depth) can be stored. The benefit of this method is that a simple calculation can be used to find a child or a parent of a node. However, often a tree (also the context tree of the CTW algorithm) is not fully extended. This results in a very sparse array and thus a large waste of memory.

- Dynamically allocating memory space for the data of each node. The benefit is that memory space is only required for nodes that are actually created, but the drawback is that extra memory is required to store pointers to the children (and if needed to the parent) of the node. Especially, when a large number of nodes have to be allocated with a small amount of data, like in the CTW algorithm, these pointers cause a huge overhead.

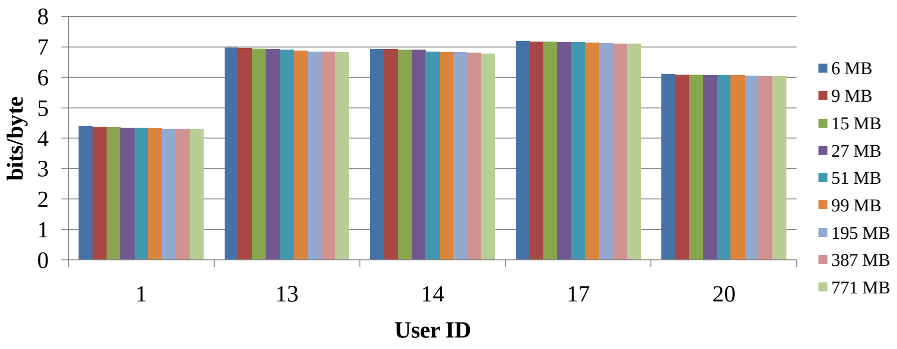
- The most efficient way of storing a data structure for statistical compression algorithms is by using *hashtables*. With hashtables, an array is used in which a fixed number of nodes can be placed. The idea is that every node is placed on a pseudo-random index in this array. The index is calculated from a certain key parameter with the so called hashing function. Hashtables provide an efficient data structure for storing the data required by statistical data compression algorithms.

In CTW algorithm, the hashing function is chosen as a pseudo-random function, such that the result of the hashing function is the same given the same key value. It is possible that different keys result in the same index; this is called a collision. To detect such a collision it is important that there is some data stored in each node which makes it possible to recognize if the right node has been found. If it is not the right node, the hashing algorithm can perform a second try, adding a certain offset to the index value and checking if the new location in the array contains the right node.

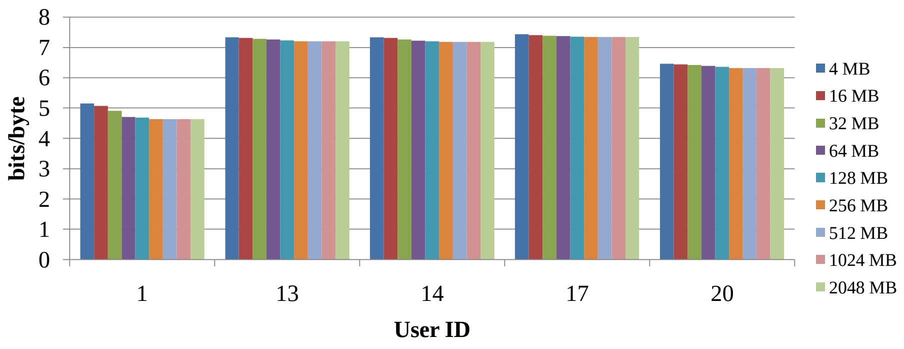
In LPAQ algorithm, two thirds of the memory used by the algorithm is used for a hashtable which maintains the order 1 through order 6 statistical models. The hash function used in the algorithm is a collision-free permutation, consisting of multiplying the input by a large odd number and rotations.

Therefore, hashtables perform a critical role in practical statistical compression methods and the dominant portion of the physical memory is allocated to these tables. Therefore, in applications where transfer of the state of the compression algorithm is needed, one only needs to extract the hashtable. This hashtable stores all the necessary data used by the compression algorithm and transferring this hashtable is sufficient to restore the state of the compression algorithm. Clearly, the more memory allocated to the table, a higher number of nodes can be accommodated in the table which in turn results in a higher compression performance. However, the increase in the compression performance should outweigh the increase in physical memory. In Figure 39, the compression performance of LPAQ (Figure 39(a)) and CTW (Figure 39(b)) algorithms for different memory sizes is depicted for a set of five mobile user data traces. For CTW algorithm, the number of nodes in the hashtable is chosen as a parameter and each node occupies 8 Bytes of memory. As we expect, the increase in memory size results in an increase in compression performance.

In Figure 40, the average compression performance of LPAQ for different memory sizes is depicted. While more memory results in improved compression performance, the improvement per additional MB of memory decreases considerably, as shown in Figure 41. Therefore, for applications that are limited in physical memory, choosing



(a) LPAQ



(b) CTW

Figure 39: The compression performance of LPAQ (a) and CTW (b) algorithms for different memory sizes and for a set of five mobile user data traces.

lower memory in exchange for a slightly reduced compression performance can be tolerated. In particular, for wireless applications choosing the small physical memory is beneficial as the overhead of transferring the compressor state from one node to another is small.

Note that the the size of physical memory used by a statistical compression algorithm can be significantly smaller than the size of processed data. For example, LPAQ algorithm with option 2 only uses 9MB of physical memory. Therefore, one can train LPAQ with large amount of data and still transfer the sate of the compressor by sending the 9MB physical footprint of the algorithm.

The overhead of transferring the compressor is quickly compensated by the saving that the statistical compression provides. For example, consider a scenario where a client has trained a statistical compressor and moves to a new network. The client can choose to send the state of compressor to the gateway (9MB overhead). Based on the results in Table 8, the trained compressor can compress every byte to $\approx$6 bits. Hence, if the client receives more than 30MB from the gateway (a 5sec connection at 54Mbit/sec speed), the benefits of memory-assisted compression (realized using a trained compressor) outweighs the overhead of sending the state of the compressor.

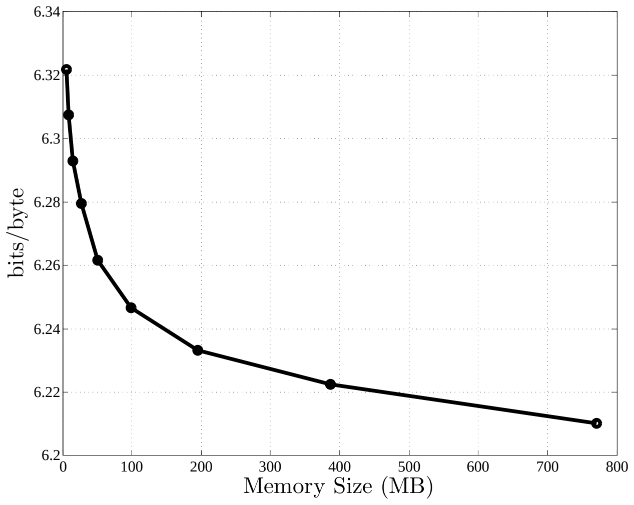


Figure 40: The average compression performance of LPAQ for different memory sizes.

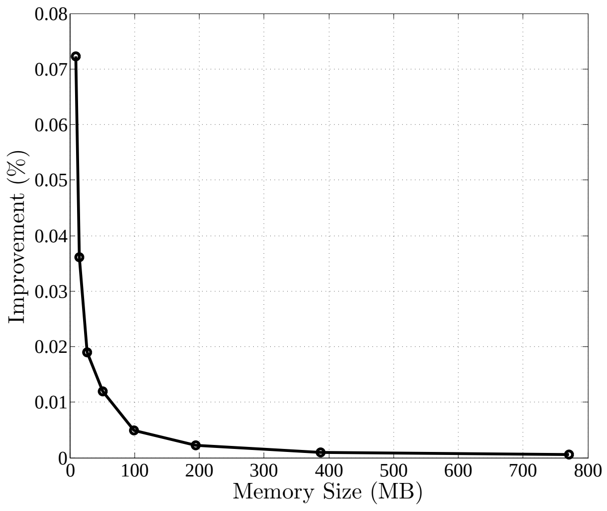


Figure 41: The percent improvement in compression performance per additional MB of memory.

CHAPTER X

CONCLUSION

In this dissertation the theoretical and practical aspects of redundancy elimination methods in data networks was investigated. Redundancy elimination provides a powerful technique to improve the efficiency of network links in the face of redundant data. A broad range of issues was investigated in the dissertation ranging from establishing the concept of memory-assisted compression and the design algorithms to application of memory-assisted compression in networks, i.e., network compression, in both wired and wireless networks. In the following, the contribution of the thesis are summarize.

In Chapter 2, the concept of memory-assisted compression was introduced. Memory-assisted compression aims at compression of individual packets using the side-information obtained from memorized data. To characterize the benefits of memory-assisted compression, two compression families, namely, statistical compression family and dictionary-based compression family were introduced and their adaption for memory-assisted compression were presented. The compression performance of these two algorithms were characterized and it was shown that statistical compression method out performs the dictionary-based method. Furthermore, the gain of memory-assisted compression was defined and characterized for both compression families.

The memory-assisted compression algorithms developed in Chapter 2 are beneficial for compression of sequences generated by a single information source. However, a single source cannot model the data traffic. Hence, a more complete model for the source, called "compound source model", was introduced in Chapter 3 and a clustering algorithm was presented which aimed at utilizing the data in the memory to better compress a new sequence from the compound source. It was shown that the clustering would result in superior performance for memory-assisted compression when the input data comprises sequences generated by various and unrelated information sources.

In the second part of the thesis the application of memory-assisted compression in wired networks was investigated. In Chapter 4 and Chapter 6, the benefits of deploying memory units that enable memory-assisted compression in wired networks were studied. In Chapter 4, the gain of network compression in the Erdős-Rényi (ER) network graph family was studied. Analogous to the memory-assisted compression gain for a single link, network-wide gain of compression was defined and studied in ER random network graphs. It was shown that the gain of network compression would depend on the number of memory units deployed in the network. In Chapter 4, it was discovered that there exists a threshold value for the number of memories deployed in a random graph below which the network-wide gain of memorization would vanish. The last problem studied in Chapter 4 was the memory placement in non-random network graphs. It was shown that optimal memory-placement is not tractable in general network graphs and the challenges involved were demonstrated by deriving the optimal memory placement on line networks.

In Chapter 6, the gain of network compression in Internet-like power-law graphs was characterized. In particular, through analysis on power-law graphs, it was demonstrated that non-vanishing network-wide gain of memorization can be obtained even when the number of memory units is a tiny fraction of the total number of nodes in the network. Furthermore, memory placement in the network poses some challenges to traditional shortest path routing algorithms, as the shortest path is not necessarily minimum cost route in networks with memory. The well-known routing algorithms like Dijkstra's algorithm, in their original form are not applicable to networks with memory. As such, in Chapter 5, the routing problem for compressed flows was considered and a modified Dijkstra's algorithm for compressed flows was presented.

In the third part of the work the application of memory-assisted compression in wireless networks was studied. In Chapter 7, a novel approach was proposed to leverage the redundancy in network data for reducing network flows by employing network compression techniques via overhearing memory units deployed as helpers in a wireless network. Each memory-enabled helper overhears the data packets previously sent by the wireless gateway to various mobile clients within its coverage and uses them toward forming a model about the content of the packets from the traffic. The resulting model is then used as side information by the wireless network compression module in a two-part code with asymmetric cost (where the helper-client link is far less costly than the server-client link). In particular, we explored the benefits of *two-part coding with asymmetric cost* in the last hop wireless links, from the wireless gateway to the mobile clients. It was shown through computer simulations that the helper nodes bring throughput enhancement and gateway off-loading which would be of particular interest for WiFi and cellular networks.

The next problem studied in this part of the thesis was the impact of loss on memory-assisted compression. Packet loss is a common feature of wireless environment. In Chapter 8, the performance of memory-assisted compression when the source model at the encoder and the decoder does not match was theoretically investigated. Furthermore, a sequential code design was presented for memory-assisted compression with mismatched side information.

The last chapter of the thesis studies the interplay between memory-assisted compression and de-duplication algorithms when used in tandem.

10.1 Suggestions for Future Research

This dissertation has opened up many interesting theoretical and practical research possibilities in memory-assisted compression in both wired and wireless networks. In the following, some of the open problems and future research directions are listed.

- Approximation algorithms for memory placement in a general network graph.
- Extension to multiple correlated sources.
- Fundamental limits of compression when the encoders do not communicate.
- Code design for memory units in the presence of multiple sources.
- Development of effective memory-assisted compression algorithms capable of working on high-speed data streams, and accommodating concurrent applications.
- Collecting a data set from an Internet router and performing memory-assisted compression and clustering on the data.
- Prototyping a modular router using Click Modular Router capable of memory-assisted compression.

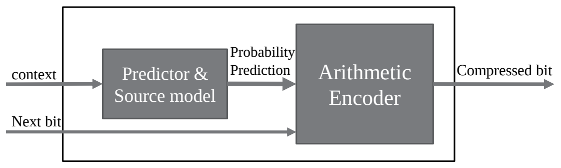
APPENDIX A

INTERNALS OF PAQ COMPRESSION ALGORITHM

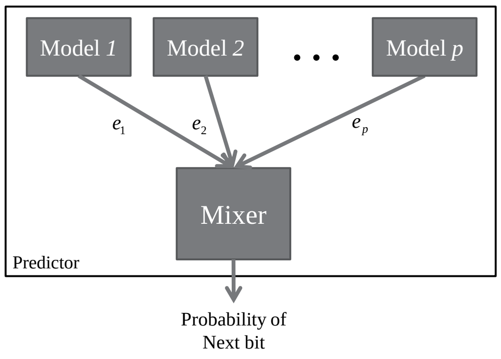
Figure 42, depicts the schematic of statistical compressor. As previously discussed in Section 2.3, the statistical compressor consists of two parts: a predictor and an entropy coder, as shown in Figure 42(a). Let $x^n = (x_1, \ldots, x_n)$, $x_t \in \mathcal{A}$ be the input sequence. The compressor processes x^n sequentially such that at t-th step the predictor emits the probability distribution of the next symbol, denoted by $\mathbf{P}[x_t|x_1^{t-1}]$, based on the already processed sequence $x_1^{t-1} = x_1, \ldots, x_{t-1}$. Given the probability distribution, the entropy coder maps x_t to a codeword of a length close to $\log \mathbf{P}[x_t|x_1^{t-1}]$ bits. In practice, Arithmetic Coding [50, 65] is used which closely approximates the ideal code length and enjoys theoretical proof on asymptotic optimality. Each model in predictor issues an estimate of the probability distribution of the next symbol. Let $e_1, \ldots, e_p$ be the estimate of models $M_1, \ldots, M_p$, respectively. The mixer, at every step, combines these estimates and emits a single prediction regarding the probability of the next symbol.

The outcome of the mixer determines the performance of the algorithm. It is common in Information Theory to assume that the source generating x^n follows a probability distribution not known to the compression algorithm. Let $\mathbf{Q}$ be the source probability distribution. Moreover, assume that the probability distribution predicted by mixer is $\mathbf{P}$. The expected length of the compressed x_t is

$$\begin{aligned} \sum_{x_t \in \mathcal{A}} \mathbf{Q}[x_t|x_1^{t-1}] \log \frac{1}{\mathbf{P}[x_t|x_1^{t-1}]} &= H(\mathbf{Q}) \\ + \sum_{x_t \in \mathcal{A}} \mathbf{Q}[x_t|x_1^{t-1}] &\left(\log \frac{1}{\mathbf{P}[x_t|x_1^{t-1}]} - \log \frac{1}{\mathbf{Q}[x_t|x_1^{t-1}]} \right), \end{aligned} \tag{65}$$



(a) Two-part design



(b) Predictor Design

Figure 42: (a) The two-part design of statistical compression algorithms composed of a predictor module and an arithmetic coding module. The predictor maintains a model for the source to use for prediction of probability of the next bit. (b) The internal design of the predictor.

where $H(\mathbf{Q})$ is the source entropy. Note that if mixer can exactly predict the source probability, i.e. $\mathbf{P} = \mathbf{Q}$, the second term on the right hand side of (65) evaluates to zero (no compression penalty) and theoretically the statistical coder is optimal for every symbol it compresses. However, the source probability is not known to the algorithm and the predictor is designed to minimize the penalty term in (65).

The minimization is performed by assigning a weight w_i to every model in the predictor and then combine the estimates of the models using the weights. Below, we discuss the implementation details of LPAQ which is the adaptation of PAQ, the most effective compression algorithm known today which holds the best compression performance in various areas.

LPAQ is a “lite” version of PAQ, about 30 times faster than PAQ8 [53] at the cost of some compression (but similar to high-end PPM compressors [78, 23]). The input sequence is processed sequentially and bit-wise. It follows the two part design discussed in Section 2.3. The predictor in LPAQ employs seven models: k-gram Markov models of orders 1, 2, 3, 4, 6, and a "match" model, which predicts the next bit in the last matching context. The independent bit probability predictions of the seven models are combined by a mixer, then arithmetic coded. The k-gram Markov models consist of the last k whole bytes plus any of the 0 to 7 previously coded bits of the current byte starting with the most significant bit.

PAQ mixer works with a binary alphabet and emits the probability of the next bit being 1. The estimates are geometrically weighted [55] and combined as

$$\mathbf{P}[1|x_1^{t-1}] = \frac{\prod_{i=1}^{p} e_i^{w_i(t-1)}}{\prod_{i=1}^{p}(1-e_i)^{w_i(t-1)} + \prod_{i=1}^{p} e_i^{w_i(t-1)}}. \tag{66}$$

The weights are then updated as follows:

$$w_i(k) = w_i(k-1) + \alpha(x_t - \mathbf{P}[1|x_1^{t-1}])\mathrm{st}(e_i), \tag{67}$$

where $\mathrm{st}(a) = \ln \frac{a}{1-a}$. It can be verified that (67) is an instance of iterative gradient descent [55, 17], where α is the update constant in any step. In LPAQ implementation $\alpha = 0.002$ is chosen. When the input sequence is stationary, the weights converge. It has been demonstrated in [55] that the mixing in (66) and (67) would result in the minimum compression penalty as defined in (65).

Here, we note that in the actual implementation, to achieve this mixing, the estimates e_i are first stretched st(.), and then the output is computed as

$$\mathbf{P}[1|x_1^{t-1}] = \mathrm{sq}\left(\sum_{i=1}^{p} \mathrm{st}(e_i)w_i(t-1)\right),$$

where $\mathrm{sq}(a) = 1/(1+e^{-a})$.

REFERENCES

[1] http://mattmahoney.net/dc/fv.zip.

[2] "Calgary Corpus." http://corpus.canterbury.ac.nz/descriptions/#calgary.

[3] "Wireshark Packet Analyser." http://www.wireshark.org/.

[4] Albert, R., Jeong, H., and Barabasi, A.-L., "Internet: Diameter of the world-wide web," *Nature*, vol. 401, pp. 130–131, 1999.

[5] Alon, N. and Spencer, J., *The Probabilistic Method–3rd edition.* John Wiley & Sons, USA, 2008.

[6] Anand, A., Gupta, A., Akella, A., Seshan, S., and Shenker, S., "Packet caches on routers: the implications of universal redundant traffic elimination," *SIGCOMM*, vol. 38, pp. 219–230, 2008.

[7] Anand, A., Muthukrishnan, C., Akella, A., and Ramjee, R., "Redundancy in network traffic: findings and implications," in *SIGMETRICS '09: Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, (New York, NY, USA), pp. 37–48, ACM, 2009.

[8] Anand, A., Sekar, V., and Akella, A., "SmartRE: an architecture for coordinated network-wide redundancy elimination," *SIGCOMM*, vol. 39, no. 4, pp. 87–98, 2009.

[9] Atteson, K., "The asymptotic redundancy of Bayes rules for Markov chains," *IEEE Trans. Info. Theory*, vol. 45, pp. 2104 –2109, September 1999.

[10] Barron, A., Rissanen, J., and Yu, B., "The minimum description length principle in coding and modeling," *IEEE Trans. Info. Theory*, vol. 44, pp. 2743 –2760, Oct. 1998.

[11] Barron, A. R. and Cover, T. M., "Minimum complexity density estimation," *IEEE Trans. Info. Theory*, vol. 37, no. 4, pp. 1034–1054, 1991.

[12] Beirami, A. and Fekri, F., "Results on the redundancy of universal compression for finite-length sequences," in *IEEE Intl. Symp. Info. Theory (ISIT)*, pp. 1504–1508, Jul 31-Aug 5 2011.

[13] Beirami, A. and Fekri, F., "Memory-assisted universal source coding," in *2012 Data Compression Conference (DCC)*, (Snowbird, Utah), p. 392, 2012.

[14] Beirami, A. and Fekri, F., "On lossless universal compression of distributed identical sources," in *IEEE Intl. Symp. Info. Theory (ISIT)*, (Boston, MA), pp. 561–565, 2012.

[15] Beirami, A., Sardari, M., and Fekri, F., "Results on the fundamental gain of memory-assisted universal source coding," in *2012 IEEE International Symposium on Information Theory (ISIT '2012)*, pp. 1092–1096, July 2012.

[16] Bentley, J. and McIlroy, D., "Data compression using long common strings," in *Data Compression Conference, 1999. Proceedings. DCC '99*, pp. 287–295, 1999.

[17] Bertsekas, D. P., *Nonlinear Programming.* Athena Scientific, 1999.

[18] Bishop, C. M., *Pattern recognition and machine learning.* Springer, 2006.

[19] Blei, D. M. and Jordan, M. I., "Variational inference for dirichlet process mixtures," *Bayesian Analysis*, pp. 121–144, 2006.

[20] Bloom, C., "LZP: a new data compression algorithm," in *Data Compression Conference. DCC '96. Proceedings*, 1996.

[21] Brady, A. and Cowen, L., "Compact routing on power law graphs with additive stretch," in *ALENEX*, 2006.

[22] Broder, A., Kumar, R., Maghoul, F., Raghavan, P., Rajagopalan, S., Stata, R., Tompkins, A., and Wiener, J., "Graph structure in the web," in *Proceedings of the WWW9 Conference*, pp. 309–320, 2000.

[23] Bunton, S., *On-Line Stochastic Processes in Data Compression.* PhD thesis, University of Washington, 1996.

[24] Cam, L. L. and Yang, G. L., *Asymptotics in Statistics: Some Basic Concepts.* Springer, 2000.

[25] Chan, M. C. and Woo, T. Y. C., "Cache-based compaction: A new technique for optimizing web transfer," in *Infocom '99: Proceedings of IEEE INFOCOM*, (New York, NY, USA), ACM, 1999.

[26] Chandrasekhar, V., Andrews, J., and Gatherer, A., "Femtocell networks: a survey," *IEEE Comm. Magazine*, vol. 46, no. 9, pp. 59–67, 2008.

[27] Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. E., "Bigtable: a distributed storage system for structured data," in *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*, (Berkeley, CA, USA), pp. 205–218, USENIX Association, 2006.

[28] Chung, F. and Lu, L., *Complex Graphs and Networks.* American Mathematical Society, 2006.

[29] Cilibrasi, R. and Vitanyi, P., "Clustering by compression," *IEEE Transactions on Information Theory*, vol. 51, pp. 1523 – 1545, April 2005.

[30] Clarke, B. and Barron, A., "Information-theoretic asymptotics of bayes methods," *IEEE Trans. Info. Theory*, vol. 36, pp. 453 –471, May 1990.

[31] Cooper, B. F. and Garcia-Molina, H., "Peer-to-peer data trading to preserve information," *ACM Trans. Inf. Syst.*, vol. 20, no. 2, pp. 133–170, 2002.

[32] Cover, T. M. and Thomas, J. A., *Elements of Information Theory.* John Wiley and sons, 2006.

[33] Erdős, P. and Rényi, A., "On random graphs. I.," *Publicationes Mathematicae*, pp. 290–297, 1959.

[34] Faloutsos, M., Faloutsos, P., and Faloutsos, C., "On power-law relationships of the internet topology," in *SIGCOMM*, pp. 251–262, 1999.

[35] Ferguson, T., "A bayesian analysis of some nonparametric problems," *The Annals of Statistics*, pp. 209–230, 1973.

[36] Ghahramani, Z. and Beal, M., "Propagation algorithms for variational bayesian learning," *Advances in Neural Information Processing Systems*, pp. 507–513, 2001.

[37] Golrezaei, N., Molisch, A. F., Dimakis, A. G., and Caire, G., "Femtocaching and device-to-device collaboration:a new architecture for wireless video distribution," *IEEE Comm. Magazine*, to appear.

[38] Grunwald, P. D., *The minimum description length principle.* The MIT Press, 2007.

[39] Gunderson, S., "Snappy." `http://code.google.com/p/snappy/`.

[40] Hidayat, A., "Fastlz." `http://www.fastlz.org`.

[41] Hsiang-Shen, S., Gember, A., Anand, A., and Akella., A., "Refactoring content overhearing to improve wireless performance," in *MobiCom*, (Las Vegas, NV), 2011.

[42] Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N., and Braynard, R., "Networking named content," in *Proceedings of the 5th ACM CoNEXT*, pp. 1–12, 2009.

[43] Karp, R. M. and Rabin, M. O., "Efficient randomized pattern-matching algorithms," *IBM Journal of Research and Development*, pp. 249–260, 1987.

[44] Knoll, B. and de Freitas, N., "A machine learning perspective on predictive coding with paq8," in *Data Compression Conference (DCC)*, (Snowbird, Utah), pp. 377–386, 2012.

[45] Kontkanen, P. and Myllymaki, P., "An empirical comparison of NML clustering algorithms," in *International Conference on Information Theory and Statistical Learning*, 2008.

[46] Kontkanen, P., Myllymak, P., Buntine, W., Rissanen, J., and Tirri, H., "An MDL framework for data clustering," tech report, Helsinki Institute for Information Technology HIIT, 2004.

[47] Koponen, T., Chawla, M., Chun, B.-G., Ermolinskiy, A., Kim, K. H., Shenker, S., and Stoica, I., "A data-oriented (and beyond) network architecture," in *SIGCOMM*, pp. 181–192, 2007.

[48] Korn, D., MacDonald, J., Mogul, J., and Vo, K., "The vcdiff generic differencing and compression data format," *RFC 3284*, 2002.

[49] Krishnan, P., Raz, D., and Shavitt, Y., "The cache location problem," *IEEE/ACM Transactions on Networking*, vol. 8, pp. 568–582, 2000.

[50] Langdon Jr., G. G., "An Introduction to Arithmetic Coding," *IBM J. Res. Develop.*, vol. 28, pp. 135–149, March 1984.

[51] Lenhardt, R. and Alakuijala, J., "Gipfeli - high speed compression algorithm," in *2012 Data Compression Conference (DCC)*, (Snowbird, Utah), pp. 109–118, 2012.

[52] Lumezanu, C., Guo, K., Spring, N., and Bhattacharjee, B., "The effect of packet loss on redundancy elimination in cellular wireless networks," in *Internet Measurement Conference*, 2010.

[53] Mahoney, M., "Adaptive weighing of context models for lossless data compression," tech. rep., Florida Tech., 2005.

[54] Manber, U., "Finding similar files in a large file system," in *Proceedings of the USENIX Winter Technical Conference*, 1994.

[55] Mattern, C., "Mixing strategies in data compression," in *Data Compression Conference (DCC)*, (Snowbird, Utah), 2012.

[56] Merhav, N. and Feder, M., "A strong version of the redundancy-capacity theorem of universal coding," *IEEE Trans. Info. Theory*, vol. 41, pp. 714 –722, May 1995.

[57] Neal, R. M., "Markov chain sampling methods for dirichlet process mixture models," *Journal of Computational and Graphical Statistics*, pp. 249–265, 2000.

[58] "The Network Simulator NS-2." `http://www.isi.edu/nsnam/ns/`.

[59] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E., "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[60] Rabin, M. O., "Fingerprinting by random polynomials," tech report, tr-cse-03-01, Harvard University, 2007.

[61] Rasmussen, C. E., "The infinite gaussian mixture model," *Advances in Neural Information Processing Systems*, pp. 554–560, 1999.

[62] Reinhold, L. M., "Quicklz." `http://www.quicklz.com`.

[63] Rhea, S. C., Liang, K., and Brewer, E., "Value-based web caching," in *Proceedings of the 12th international conference on World Wide Web*, WWW '03, (New York, NY, USA), pp. 619–628, ACM, 2003.

[64] Rissanen, J., "Strong optimality of the normalized ML models as universal codes and information in data," *IEEE Trans. Info. Theory*, vol. 47, pp. 1712 –1717, July 2001.

[65] Rissanen, J. J., "Generalized Kraft Inequality and Arithmetic Coding," *IBM J. Res. Develop.*, pp. 198–203, May 1976.

[66] Rissanen, J., "Fisher information and stochastic complexity," *IEEE Trans. Info. Theory*, vol. 42, pp. 40 –47, Jan. 1996.

[67] Saleh, O. and Hefeeda, M., "Modeling and caching of peer-to-peer traffic," in *ICNP '06: Proceedings of the Proceedings of the 2006 IEEE International Conference on Network Protocols*, (Washington, DC, USA), pp. 249–258, IEEE Computer Society, 2006.

[68] Sanadhya, S., Sivakumar, R., Kim, K.-H., Congdon, P., Lakshmanan, S., and Singh, J., "Asymmetric caching: Improved deduplication for mobile devices," in *Proceedings of the ACM MOBICOM 2012 conference*, ACM, 2012.

[69] Sardari, M., Beirami, A., and Fekri, F., "Wireless network compression via memory-enabled overhearing helpers," submitted to *IEEE Transactions on Wireless Communications.*

[70] Sardari, M., Beirami, A., and Fekri, F., "On the network-wide gain of memory-assisted source coding," in *2011 IEEE Information Theory Workshop (ITW)*, pp. 476–480, October 2011.

[71] Sardari, M., Beirami, A., and Fekri, F., "Memory-assisted universal compression of network flows," in *IEEE INFOCOM*, (Orlando, FL), pp. 91–99, March 2012.

[72] Sardari, M., Beirami, A., and Fekri, F., "Wireless network compression: Code design and trade offs," in *Information Theory and Applications Workshop (ITA)*, (San Diego, CA), 2013.

[73] Sardari, M., Beirami, A., Zou, J., and Fekri, F., "Content-aware network data compression using joint memorization and clustering," in *IEEE INFOCOM*, (Turin, Italy), April 2013.

[74] Sardari, M., Beirami, A., and Fekri, F., "Memory placement in network compression: Line and grid topologies," in *International Symposium on Information Theory and its Applications (ISITA)*, (Honolulu, HI, USA), Oct. 2012.

[75] Sartipi, M. and Fekri, F., "Distributed source coding using short to moderate length rate-compatible LDPC codes: the entire Slepian-Wolf rate region," *IEEE Transactions on Communications*, vol. 56, pp. 400–411, March 2008.

[76] Shamir, G. I., Tjalkens, T. J., and Willems, F. M. J., "Low-complexity sequential probability estimation and universal compression for binary sequences with constrained distributions," in *IEEE Intl. Symp. Info. Theory (ISIT)*, (Toronto, Canada), pp. 995–999, 2008.

[77] Shannon, C. E., "A Mathematical Theory of Communication," *The Bell System Technical Journal*, vol. 27, pp. 379–423, 623–656, Jul., Oct. 1948.

[78] Shkarin, D., "PPM: one step to practicality," in *Data Compression Conference*, vol. 12, 2002.

[79] Slepian, D. and Wolf, J. K., "Noiseless coding of correlated information sources," *IEEE Trans. Info. Theory*, vol. 19, pp. 471–480, 1973.

[80] Spring, N. T. and Wetherall, D., "A protocol-independent technique for eliminating redundant network traffic," *SIGCOMM*, vol. 30, no. 4, pp. 87–95, 2000.

[81] Sundaram, R., Rao, S. B., Miller, G. L., Canfield, T. K., and Bornstein, C. F., *Optimal route selection in a content delivery network*. United States Patent 7274658.

[82] Teh, Y. W., "Dirichlet processes," *Machine Learning Summer School–Tutorial and Practical Course*, 2007.

[83] Wainwright, M. J. and Jordan, M. I., "Graphical models, exponential families, and variational inference," in *Foundations and Trends in Machine Learning*, pp. 1–305, NOW, 2008.

[84] Willems, F., "The context-tree weighting method: extensions," *IEEE Trans. Info. Theory*, vol. 44, pp. 792–798, March 1998.

[85] Willems, F., Shtarkov, Y., and Tjalkens, T., "The context-tree weighting method: basic properties," *IEEE Trans. Info. Theory*, vol. 41, pp. 653–664, May 1995.

[86] Zhuang, Z., Kakumanu, S., Jeong, Y., Sivakumar, R., and Velayutham, A., "On the impact of mobile hosts in peer-to-peer data networks," in *ICDCS '08: Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, (Beijing, China), IEEE Computer Society, 2008.

[87] Zhuang, Z., Tsao, C.-L., and Sivakumar, R., "Curing the amnesia: Network memory for the internet," tech. report, Georgia Institute of Technology, 2009.

[88] Ziv, J. and Lempel, A., "A universal algorithm for sequential data compression," *IEEE Trans. Info. Theory*, vol. 23, pp. 337–343, May 1977.

VITA

Mohsen Sardari received his B.Sc. degree in Electrical Engineering from Sharif University of Technology, Tehran, Iran, in 2007. He received his M.S.E.C.E. degree from the School of Electrical and Computer Engineering (ECE), Georgia Institute of Technology, Atlanta, GA, in 2010. He is currently a Graduate Research Assistant in the Information Processing, Communications and Security Research Laboratory and pursuing his Ph.D. degree at the School of ECE, Georgia Institute of Technology, Atlanta, GA. His current research interests include redundancy elimination, data compression, and wireless networks. Mohsen Sardari is the recipient of 2013 Outstanding Service Award from the Center of Signal and Information Processing (CSIP) at Georgia Tech.