Packet-Level Network Compression: Realization and Scaling of the Network-Wide Benefits

Ahmad Beirami, *Student Member, IEEE*, Mohsen Sardari, and Faramarz Fekri, *Senior Member, IEEE*

***Abstract*—The existence of considerable amount of redundancy in the Internet traffic at the packet level has stimulated the deployment of packet-level redundancy elimination techniques within the network by enabling network nodes to memorize data packets. Redundancy elimination results in traffic reduction which in turn improves the efficiency of network links. In this paper, the concept of network compression is introduced that aspires to exploit the statistical correlation beyond removing large duplicate strings from the flow to better suppress redundancy. In the first part of the paper, we introduce "memory-assisted compression," which utilizes the memorized content within the network to 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. Using simulations on data gathered from real network traces, we show that memory-assisted compression can result in significant traffic reduction. In the second part of the paper, we study the scaling of the average network-wide benefits of memory-assisted compression. We discuss routing and memory placement problems in network for the reduction of overall traffic. We derive a closed-form expression for the scaling of the gain in Erdős-Rényi random network graphs, where obtain a threshold value for the number of memories deployed in a random graph beyond which network-wide benefits start to shine. Finally, the network-wide benefits are studied on Internet-like scale-free networks. We show that non-vanishing network compression gain is obtained even when only a tiny fraction of the total number of nodes in the network are memory-enabled.**

***Index Terms*—Dijkstra's algorithm, Erdős-Rényi random graphs, memory-assisted compression, network memory, random power-law graph, redundancy elimination.**

I. Introduction

MASSIVE amount of data is daily produced and transmitted through various networks. A very high fraction of the cost of dealing with such massive data is associated with the transmission. Hence, any mechanism that can provide traffic reduction would result in huge cost benefits. Several studies have examined real-world network traffic and concluded the presence of considerable amounts of redundancy in the traffic data [1]–[6]. 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 [7], [8]. However, several experiments confirm that the caching approaches, which take place at the application layer, do not efficiently leverage the network traffic redundancy which exists mostly at the packet level [2], [5]. 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 in the network layer (i.e., layer 3) [2], [5], [9], [10], 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 so high that even a simple de-duplication method (i.e., removing the repeated segments of the packets) can provide considerable bandwidth savings. Motivated by these benefits, in this work, packet-level redundancy elimination is investigated from an information-theoretic point of view.

Information theory has already established the fundamental limit in the compression of infinite-length sequences for the class of universal schemes [11]–[13]. Entropy is the fundamental limit (also called Shannon 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 [14], [15], there is a significant penalty, i.e., gap from the asymptotic limit, when finite-length sequences are compressed under a universal scheme. More precisely, there is no hope of developing source coding algorithms that can compress short sequences to their entropy [15]. Also, in [15], [16], we showed that the compression of small network packets requires more than 100% compression overhead, beyond the Shannon limit. There is no way to get around this limit in the absence of side information.

It is natural to ask whether it is possible to improve compression rates by taking advantage of side information about the source provided by the memorized sequences from previous

Manuscript received April 29, 2014; revised November 18, 2014; accepted March 27, 2015; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor A. Markopoulou. Date of publication May 22, 2015; date of current version June 14, 2016. This material is based upon work supported by the National Science Foundation under Grant No. CNS-1017234. This paper was presented in part at the 2011 IEEE Information Theory Workshop (ITW 2011) and the 2012 IEEE International Conference on Computer Communications (INFOCOM 2012).

A. Beirami was with Georgia Institute of Technology, Atlanta, GA 30332 USA, and is currently jointly affiliated with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA, and the Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: ahmad.beirami@duke.edu; beirami@mit.edu).

M. Sardari was with Georgia Institute of Technology, Atlanta, GA 30332 USA, and is now with Electronic Arts, Inc., Redwood City, CA 94065 USA (e-mail: sardari@gatech.edu).

F. Fekri is with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: fekri@ece.gatech.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2015.2418296

traffic. In the finite block length regime of practical interest, it was demonstrated that by adding only 4 MB of memory at the router it is possible to learn enough about a stationary source to overcome the fundamental limits of universal compression in the finite-length regime and approach the Shannon limit [16]. This is referred to as memory-assisted universal compression, where a side information sequence of length m (that can be thought of as the aggregate of the previous traffic stored in the network node) is obtained from the information-generating source. This length m sequence that is stored both at the encoder (i.e., the source) and the decoder (i.e., the memory-enabled network node) is called "memory" and conveys important information about the source statistics.

Even if the gain of memory-assisted compression is justified on a link, it remains an open problem how such a gain would scale if memory-assisted compression is adopted in the Internet on the router level. Several studies have inferred that the Internet (at the router level) can be very well modeled using a scale-free network [17]–[21]. In particular, these studies provide evidence that although the Internet is growing dynamically, its properties can be well modeled through a stationary state described using the scale-free network model. Therefore, the end goal of this paper would be to understand the scaling of the network compression gain on Internet-like scale-free networks.

In the first part of this paper (through Section IV), we discuss practical algorithms for implementation of *memory-assisted compression* and confirm the non-trivial gain of memory-assisted compression on data gathered from real network traces. This is the achievable improvement in a link between the encoder and the decoder which both have a shared side information (or memory) of the previous communication and compare against the existing redundancy elimination techniques. The memory-assisted compression is performed using both dictionary-based compression, e.g., gzip (also referred to as LZ77) [11] and statistical compression, e.g., LPAQ [22], and context tree weighting (CTW) [12]. These results provide a foundation for the network compression, which extends the idea of memory-assisted compression beyond a single link.

In the second part of this paper (from Section V), we extend our work to find achievable network-wide gain of memory-assisted compression (also referred to as network compression gain). In this context, we demonstrate that 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 requiring modification in routing. Further, we show that optimal memory placement in a network is non-trivial and vital to achieving network-wide benefits from memory deployment. To determine how the network compression gain scales with the number of memory-enabled nodes in the network, we theoretically quantify the scaling behavior of the benefits of memory-enabled nodes, Erdős-Rényi (ER) random network graphs [23]. Considering ER random network graphs simplifies the problem significantly as the memory placement problem is trivial due to the symmetry and the analysis yields to a closed-form solution. The exact analysis reveals 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 and above which it is fully accessible. Finally, network compression gain is studied in Internet-like scale-free networks (which are modeled in this paper using random power-law graphs). Through analysis on random 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.

Our contributions in this paper are summarized below.

- The concept of memory-assisted compression for redundancy elimination is introduced and its benefits are validated on real network data. In particular, memory-assisted compression gain is also defined which is the fundamental benefit that is obtained from memory in the network packet compression.
- Network compression gain is defined and optimal routing strategy and memory placement algorithms in the presence of memory-enabled nodes are presented to maximize the network compression gain when the objective is the reduction of the aggregate traffic in the entire network.
- The *average case* scaling of network compression gain is studied on Erdős-Rényi random network graphs. It is shown that even with deployment of memory-enabled nodes that scale sublinearly with the total number of nodes in the network, non-negligible benefit from network compression is achievable on the average.
- The *average case* scaling of network compression gain is studied on scale-free networks modeled using Internet-like random power-law graphs, and it is demonstrated that significant network-wide benefits are obtained when only a tiny fraction of the network nodes are memory-enabled.

The rest of the paper is organized as follows. In Section II, the related work in network traffic reduction (redundancy elimination) is reviewed. In Section III, memory-assisted universal compression is introduced. In Section IV, the benefits of memory-assisted compression for redundancy elimination are investigated through simulations on data gathered from real network traces. In Section V, the issues of extending memory-assisted compression to a network are described and the network compression gain is defined. In Section VI, the optimal routing and memory placement are investigated for maximizing the network-wide gain. In Section VII, network compression is studied for Erdős-Rényi random graphs. In Section VIII, network compression is investigated for random power-law graphs. Finally, the conclusion is given in Section IX.

II. Related Work

A. Content-Centric Networking

Recently, there has been a lot of attention regarding the efficient utilization of memory units inside network. One related line of work is the content-centric networking (CCN) [24], [25]. In CCN, content is segmented into individually addressable pieces, and these individually addressable data segments are cached in the network. However, there are several fundamental differences between our work on network compression and the previous research on CCN. 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 be

able to leverage the memory-assisted 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 [5], due to this predefined fixed granularity.

B. Redundancy Elimination Using De-Duplication

Another very related line of work considers the benefits of the deployment of packet-level redundancy elimination in the network [2]–[6], [26]. The mechanism considered for redundancy elimination is mainly based on de-duplication. The de-duplication mechanism identifies the largest chunk of data that appears in memory and replaces it with a pointer [27]. It is common for network traffic to contain large repeated blocks, e.g., traffic from users that watch the same video.

The core to de-duplication is an efficient value-based fingerprinting algorithm that is used to identify repeated chunks of data. In [28], Karp and Rabin originally presented their pattern matching algorithm for string searching; the algorithm answers whether a particular pattern sequence exists in a packet of length n by an efficient value-based fingerprinting [1], [29]. The finger printing process facilitates the de-duplication by providing an easy to compute function that can quickly lead to identification of duplicates in the packets.

De-duplication works well when the redundancy across the packets is in the form of large repeated chunks from a previously communicated packet. However, redundancy in data exists beyond simple repetitions in the form of statistical dependencies between symbols. This motivates our information-theoretic investigation of more efficient redundancy elimination algorithms based on data compression that target to leverage these statistical dependencies. We stress that the benefits of de-duplication and memory-assisted compression are complementary to each other as de-duplication provides a fast and efficient way of removing large repetitions whereas such repetitions would go mostly undetected using memory-assisted compression. On the other hand, universal compression targets to leverage statistical dependencies between symbols in a sequence that would not be well detected using the de-duplication techniques. This observation is confirmed experimentally in Section IV-B.

C. Compression (Source Coding)

While relevant, the network compression problem is different from those addressed by distributed source coding techniques (i.e., the Slepian-Wolf problem) that target multiple correlated sources sending information to the same destination [30], [31]. In the Slepian-Wolf problem, 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.

Finally, as described in the introduction, fundamental limits of finite-length packet compression using memory-assisted compression were theoretically studied in [15], [16] for a single source-destination link. This work extends the concept of memory-assisted compression to networks and aims at investigating the achievable network-wide compression benefits. Network compression requires new types of compression techniques that would take into account what is already memorized in the memory units and try to achieve the abovementioned fundamental limits laid down in [15], [16].

D. Network-Wide Issues of Redundancy Elimination

In [2], [4], Anand *et al.* discussed network-wide issues of implementing redundancy elimination techniques, where they devise redundancy elimination-aware routing techniques for ISPs. This is done for traffic engineering objectives more advanced than the conventional minimum $\mathsf{bit} \times \mathsf{hop}$ routing. In contrast, we demonstrate that even addressing the minimum $\mathsf{bit} \times \mathsf{hop}$ routing is non-trivial for memory-assisted compression and leave the extension to more advanced traffic engineering as an open future research direction. Since memory-assisted compression is complementary to the de-duplication based redundancy elimination, some of the solutions of [2] for the latter probelm could be adapted to the network compression problem as well. We stress that the goal of this paper is to investigate the average case scaling behavior of the network compression in an abstract sense as a function of the number of nodes in scale-free networks assuming that the practical implementation would be feasible.

As discussed in [4], redundancy elimination in a network-wide setting results in non-trivial deployment challenges. First, since redundancy elimination requires the intermediate nodes to operate on the incoming packets, any solution must take into account the physical constraints and limitations of the available resources to carry out such operations. As such, in Section III-C, we address the tradeoff between compression rate (performance) and compression complexity (required resources) of the developed memory-assisted compression. Another issue that needs to be addressed is that the architecture should be flexible so that network operators can choose and meet their overall network-wide goals using redundancy elimination. Although, in this paper, we only focus on overall traffic reduction as the simplest objective of network compression, more general objectives can be considered for network compression leading to different routing and memory deployment strategies. Lastly, the architecture should be designed in such a way that it can be adopted in an incremental fashion, i.e., you can equip network nodes with redundancy elimination capability one-by-one. We believe this latter issue can be addressed using the same techniques developed in [4] for redundancy elimination based on de-duplication.

III. Memory-Assisted Universal Compression

Consider an information source node S which generates content to be delivered in the form of packets to a destination (client) node $D \in \mathbf{D}$ connected to S through memory node μ, as shown in Fig. 1. Further, the client nodes in $\mathbf{D}$ request various sequences from the source over time. Let $x^n = (x_1, \ldots, x_n)$ be a sequence of length n, where each symbol x_i is from the alphabet $\mathcal{A}$. For example, for an 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

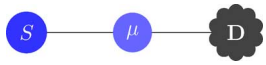


Fig. 1. The basic source S, memory μ, and destination $\mathbf{D}$ configuration, where $\mathbf{D}$ represents a set of clients receiving packets from S.

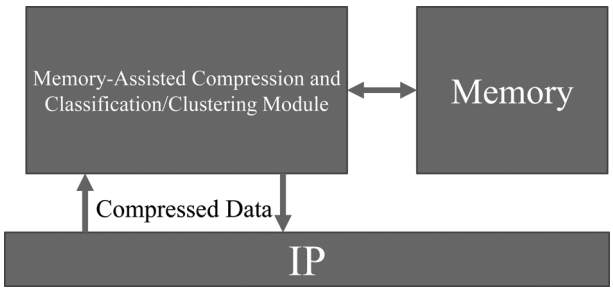


Fig. 2. Network Compression architecture which includes the classification/clustering module.

$E[l_n(X^n)]$ denote the expected length resulting from the universal compression of x^n.

Here, we consider two compression scenarios, as follows:

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 with side information (UcompM), in which the encoder (e.g., server S in Fig. 1) and the decoder (e.g., at the intermediate node μ in Fig. 1) both have access to a common side information (memory) y^m 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 client D in sub-network $\mathbf{D}$. As such, compression takes place in the source and decompression is performed in the destination. Assuming a universal compression at the source, $E[l_n(X^n)]$ 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 $\mathbf{D}$ requests a different sequence x^n (but statistically dependent) over time, the source must encode each sequence x^n independently and route through μ. As such, clearly Ucomp does not utilize the correlations across different packets. Now, consider the second scenario in which the intermediate node μ, while serving as an intermediate node for different contents destined for different clients, memorize the contents and also constructs a model for the source S. As both source S and the intermediate node μ are aware of the previous content sent to another client in $\mathbf{D}$, they can leverage this common knowledge for the better compression of the new packets sent over the $S-\mu$ portion of the path.

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 Fig. 2. Second, the intermediate node μ must decode and re-encode as the client at destination lacks memory, and hence, the client 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 (possibly) 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.

Specifically, assume that previous sequences (packets) $x^{m_1},\dots,x^{m_L}$ are sent from S to clients $D_1,\dots D_L$ in the sub-network $\mathbf{D}$ via μ. Under UcompM, the node μ constructs a model for the source S by observing the entire length $m=m_1+\cdots+m_L$ sequence. Note that forming the source model by node μ is not a passive storage of the sequences $x^{m_1},\dots,x^{m_L}$. This source model would be extracted differently for different universal compression schemes that will be used 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 of the previous traffic as side information 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. From now on, by memory size we mean the total length m of the observed sequences from the source at the memory unit. We also stress that the network compression gain only applies after the initial memorization phase in which the memory-enabled routers populate their memory with packets from previous communication.

Let $E[l_{n|m}(X^n,Y^m)]$ be the expected code length for a sequence of length n given a side information sequence Y^m of length m that is available to both the encoder and the decoder. The gain of memory-assisted compression $g\,(n,m)$ is defined as

$$g\,(n,m)\triangleq\frac{E[l_n(X^n)]}{E[l_{n|m}(X^n,Y^m)]}. \tag{1}$$

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 (i.e., Ucomp) when the encoder and the decoder have a common side information sequence of length m.

The memory-assisted compression gain has been theoretically investigated in [16] where bounds on the achievable memory-assisted compression gain are provided for stationary parametric sources. It has been shown that with a memory size of 4 MB from the same parametric source, it is possible to obtain more than two-fold gain in the compression rate of a new sequence. On the other hand, the purpose of this work is to

validate such benefits on data gathered from real traffic traces using practical algorithms. Practical compression algorithms can be divided into two categories: statistical compression methods and the dictionary-based compression methods, which are discussed in Sections III-A and III-B, respectively. In Section III-C, we present the tradeoffs between the performance and complexity of memory-assisted compression algorithms.

A. Statistical Compression Methods

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 [32]. 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 likelihood of the symbol (e.g., byte) based on the model and context (previously seen symbols).
2) Passing the estimated likelihood to the arithmetic encoder, which encodes the symbol.
3) Updating the model using the new symbol.

The decoding process is very similar to the encoding. Modern statistical compression algorithms such as Context Tree Weighting (CTW) [12], [33], Prediction by Partial Matching (PPM) [34], [35], and PAQ [22], [36] mix multiple simple models constructed sequentially to achieve better compression.

1) Context Tree Weighting (CTW): A simple yet effective predictor can be constructed using tree models; Context Tree Weighting (CTW) algorithm is a well-known example of this approach [12], [33]. CTW is used in part of the experiments in this paper. 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 leaf 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 Fig. 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 (such as the one in TCP/IP) suffices for the context synchronization.

2) Lite PAQ (LPAQ): LPAQ is a "lite" version of PAQ, about 30 times faster than PAQ8 [22] at the cost of some compression (but similar to high-end PPM compressors [34], [35]). The input sequence is processed sequentially and bit-wise. It follows the two part design discussed in Section III-A. 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 [37] and combined. It can be verified that PAQ solves an instance of iterative gradient descent [37], [38]. When the input sequence is stationary, the weights can be shown to converge to the true statistics of the source.

B. Dictionary-Based Compression Methods

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.

Dictionary-based compression algorithms have root in the seminal work of Lempel and Ziv [11], [39]. This algorithm is based on a dynamically encoded dictionary that replaces a continuous stream of characters with codes. The symbols represented by the codes are stored in memory in a dictionary-style list. Dictionary-based algorithms are widely used in practice and can be implemented efficiently and fast. However, the compression performance of these algorithms is considerably worse than a properly implemented statistical compression method.

Gzip (LZ77): In this work, we use memory-assisted gzip, which is implemented 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 gzip (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 gzip (LZ77) decoding algorithm.

C. Compression Complexity

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 all of which can be considered variants of gzip, such as [40]–[43]. These algorithms are the key part of some of the massive parallel computation systems, for example, Snappy [41] 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 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 1 GB of the English Wikipedia using Snappy [41], and Gipfeli [40] has resulted in 530 MB (in 2.8 sec) and 410 MB (in 4.3 sec), respectively. However, the implementation of gzip that we experimented on would compress the same input to 320 MB (in 41.7 sec).[1] In contrast, PAQ8, CTW, and lite PAQ compress the 1 GB English text input to 134 MB (in $\approx$ 30 ksec), 211 MB (in $\approx$ 13 ks), and 164 MB (in $\approx$ 1 ksec), respectively. We note that improving compression speed and reducing the complexity of compression while maintaining acceptable compression rate is the subject of active research in the compression community (cf. [44] where the authors improve both compression performance and compression speed by using parallel compression). In conclusion, the high-speed compression algorithms are suitable where communication throughput is high and processing power is limited, e.g., 128 MB/sec for Ethernet 1 Gigabit/sec connection. The high-performance statistical compression algorithms are more suitable for in a link where the communication speed is lower (6.5 MB/sec for 802.11g) and higher compression rates are desirable.

IV. Memory-Assisted Compression Gain on Real Network Traces

A. CNN Website Test Scenario

In this section, we first demonstrate the shortcomings of universal compression methods (without side information) for network packet compression. The data used in this experiment is downloaded from CNN website (which is mostly text and script files). To capture the packet, we used *wget* and *wireshark* [45] open-source packet analyzer together and stored the IP packets. We captured more than 18,000 data packets from the website. All packets have the same size of 1,434 bytes. In the first part of the experiment, we concatenated the packets to derive varying size super-packets and applied gzip (LZ77) and CTW on them.

As shown in Fig. 3, a modest compression performance can be achieved by compression of a packet when the super-packet length n is small to moderate size. For example, for a data packet of length $n = 1$ kB, the compression rate is about 5 bits per byte. Note that the uncompressed packet requires 8 bits per byte for representation. Observe that as the packet length n increases (here we have concatenated several packets payloads to achieve varying size packets), the compression performance improves. For very long sequences, the compression rate is about 0.5 bits per byte. In other words, comparing the compression performance between $n = 1$ kB and $n = 16$ MB, there is a penalty of factor 10 on the compression performance (i.e., 5 as opposed to 0.5). Please note that the main reason this data set is compressible by more than 10 times is that it mostly consists of text files and scripts.

Next, we applied memory-assisted versions of LZ77 and CTW on the same data packets as depicted in Fig. 4. To obtain the results in this plot, the first 4 MB worth of packets from the data is used as memory. Then, 100 sequence are chosen

[1]The execution time 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.

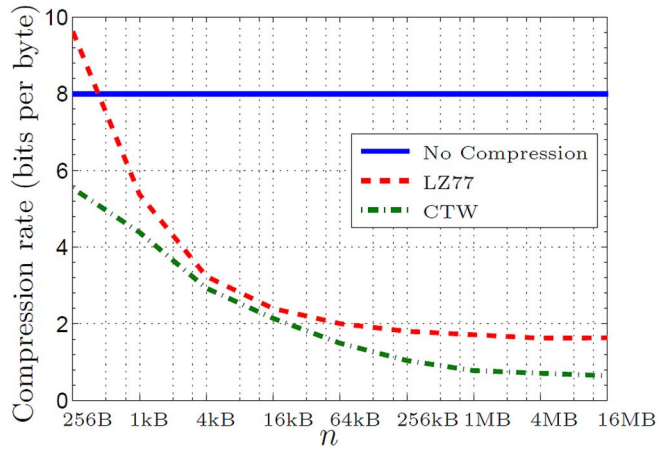


Fig. 3. The compression rate of a sample web trace (obtained from CNN web server) as a function of the sequence (i.e., packet) length, obtained using LZ77 and CTW compression algorithms.

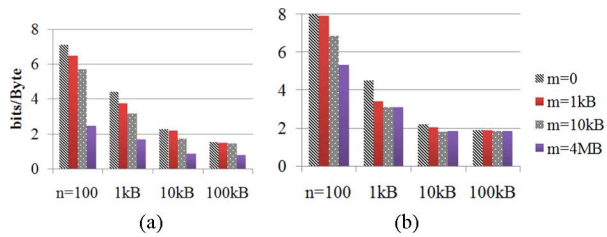


Fig. 4. The compression ratio (bits/Byte) achieved by memory-assisted universal compression algorithms. (a) Memory-assisted CTW. (b) Memory-assisted gzip (LZ77).

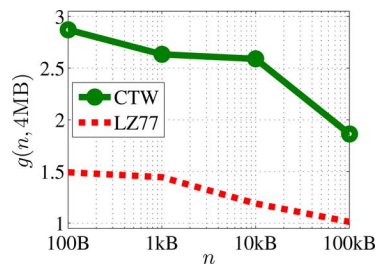


Fig. 5. The gain g of memory-assisted compression over traditional compression (Ucomp), for memory size of 4 MB for CTW and LZ compression algorithms. This gain is achieved by utilizing memory on top of the performance of the conventional compression.

from the rest of the data for compression and the average performance is reported in Fig. 4. As expected, the size of the compressed sequence decreases as memory size m increases. For example, for a data sequence of length $n = 100$B (which is obtained by manually extracting 100 B from the payload of the packet), without memory, the compressed sequence has an average length of $\approx$ 87B, while using a memory of size $m = 4$ MB, this data sequences can be compressed on average to 31 B; almost 3 times smaller. This validates the theoretical predictions in [16] about the improvements achieved using memory-assisted compression.

The actual gain of memory-assisted compression g (defined in (1)) for memory size 4 MB is depicted in Fig. 5. As can be inferred, the memory-assisted CTW (which is a statistical compression method) outperforms memory-assisted LZ77 (which is a dictionary-based method) in both the absolute size of the compressed output and also the gain of memory, i.e., the gain

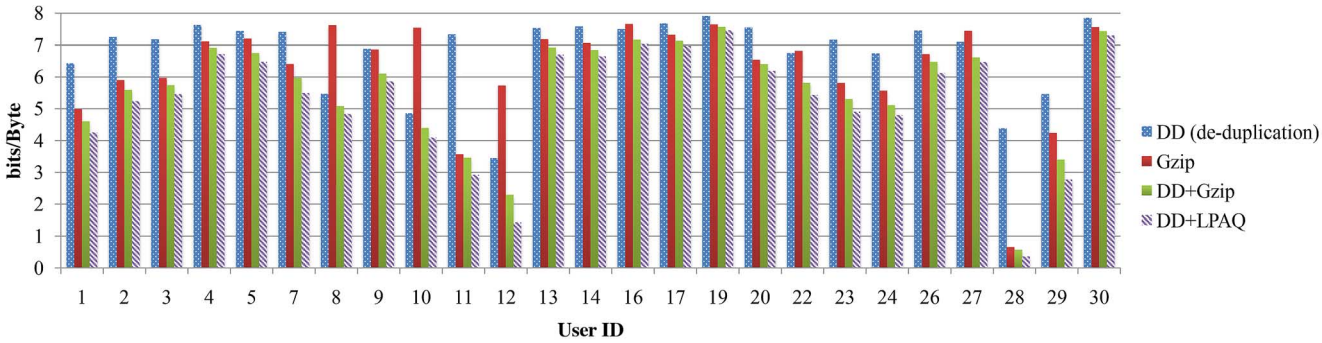


Fig. 6. Performance of various compression algorithms on the data set used in the wireless users test scenario in Section IV-B.

achieved on top of the gain of conventional compression, by utilizing memory. On the other hand, as discussed in Section III-C, the suitable compression method has to be chosen based on the specific compression complexity requirements that need to be satisfied.

B. Wireless Users Test Scenario

In this test scenario, we consider the data set used in [6]. The data set includes 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 [6].

We also used the implementation of de-duplication from [6] for the sake of comparison with memory-assisted compression. The results are presented in Fig. 6, where the first bar shows the outcome of de-duplication, denoted by DD in short, and the second bar is the compression result of the trace data using gzip algorithm. 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 1,500 bytes. Each trace of each user is processed individually and the compression is performed on the entire trace data. As expected, the performance of varies for different users. In particular, in some cases, redundancy elimination is capable of reducing the input size by half whereas in some other cases little reduction can be observed.

Although gzip slightly outperforms de-duplication on most users, it is important to note that compression-based redundancy elimination in some cases performs worse than the de-duplication, e.g., on traces of Users 10 and 12. To understand this difference in behavior, we further analyzed the data for Users 11 and 12. For User 11, gzip outperforms de-duplication and for User 12, it is the other way around. By analyzing the contents, we discovered that for User 11 long repeated sequences are scarce. Hence, the redundancy mostly exists in the form of statistical dependencies that can be captured using compression-based methods. On the other hand, the data for User 12 abounds with long duplicates that are of tens of megabytes long. In such cases, de-duplication can more efficiently eliminate the existing redundancy. This confirms that de-duplication and compression target different types of redundancies in the data.

TABLE I
SUMMARY OF THE COMPRESSION RATE (BITS/BYTE) OF VARIOUS DE-DUPLICATION AND MEMORY-ASSISTED COMPRESSION ALGORITHMS.

De-duplication (DD)	Gzip	DD+Gzip	DD+LPAQ
6.80	6.24	5.61	5.27

Next, we consider the interplay between de-duplication and memory-assisted compression As described in Section II-B, de-duplication can be used to identify and remove the long repeated chunks of data. Removing these repeated chunks before feeding the input sequence to the compression engine provides two major benefits for memory-assisted compression. First, the use of relatively fast de-duplication can speed up the high-performance but low-speed statistical data compressors. Second, the large repeated patterns would impact the predictor as it will try to learn and adapt to them while these patterns are not part of the existing statistical dependencies.

The results of applying de-duplication before memory-assisted compression are presented in Fig. 6. As can be seen, the combination consistently outperforms both de-duplication and memory-assisted compression on all traces. Further, Table I presents the average traffic reduction over all users. As can be seen, de-duplication achieves 15% traffic reduction over these users compared with 22% achieved by gzip memory-assisted compression. What is perhaps more interesting is the fact that when the two are applied in tandem, 30% traffic reduction is achieved reaffirming that the two techniques target different types of redundancy in the data. Further, as expected the best performance is attributed to de-duplication in tandem with LPAQ memory-assisted compression achieving around 34% traffic reduction on this data set.

C. Memory Requirements

To investigate the impact of the amount of physical memory on the compression performance, we derived the average compression performance of gzip, CTW, and LPAQ for different memory sizes. By increasing the memory size of gzip beyond 4 MB (which is its standard memory size), compression performance improves and gets to close to the statistical compression methods with the cost of added complexity, which scales linearly with the memory size. On the other hand, the main purpose that gzip may be adopted is when fast compression is desired. For the statistical compression methods, we discovered that in both CTW and LPAQ, less than 2% performance improvement

is achieved when the memory size is increased from 4 MB to 800 MB (on the wireless users data set). Therefore, in this paper, in most of the experiments we chose to use a memory of size 4 MB.

We stress that it is reasonable to expect that a router that is sitting in the core of the network might see traffic from several different sources (users), which might be drastically different. Therefore, one might expect that the required memory size would then become much higher for such a router. For example, when a completely uncorrelated side information sequence is used for the compression of the current sequence, there is no hope in exploiting the side information for better compression. On the other hand, in [46], we showed if the side information data is sufficiently correlated, high compression gains would be expected for a broad range of degrees of correlation. Further, in [47], we mixed the data from the wireless users (to simulate what happens to a core router that sees all the data). We observed that using only 5 models, we can very well compress any sequence from all of these sources. Hence, we believe that although 4 MB memory would probably be insufficient for a core router, the required memory size would not be unbounded either.

V. Network-Wide Gain of Memory-Assisted Compression

Thus far, we demonstrated that significant gain can be achieved using memory-assisted compression for redundancy elimination on the link level. The goal of the second part of this paper is to investigate how these benefits scale when memory-enabled nodes are deployed in a large-scale network. Specifically, the question that we are interested in answering is: "Given the memory-assisted compression gain g, and a number of memory-enabled nodes capable of performing memory-assisted compression, and their placement, what is the achievable network-wide gain?" Note that traffic traversing different paths in the network would see different gains at different time instances. On the other hand, we stress that our goal is present an average case study. Hence, in this section, we assume that each time a packet is compressed using memory-assisted compression, it experiences a gain g (on the average). Although our assumptions do not provide much information about an individual traffic packet, we can draw conclusions about the *average* traffic reduction in the entire network and such conclusions become more and more relevant as the size of the underlying large-scale network grows larger.

We represent a network 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-enabled nodes $\boldsymbol{\mu}=\{\mu_i\}_{i=1}^{M}$ chosen out of N nodes where every memory node μ_i is capable of memorizing the communication passing through it. In this paper, as the first step, we assume that the total size of memorized sequences for each μ_i is assumed to be equal to m and also assume that these nodes have similar resource constraints. Hence, we can assume that each memory unit will provide the same memory-assisted compression gain g on the link from the origin node of the flow to itself. The extension to meet resource limitations of individual nodes is left as an interesting future direction.

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 uniformly at random, i.e., each node would be selected with probability $\frac{1}{N}$ and the destination is independently selected uniformly at random as well. In this paper, as the first step, we focus on minimizing the total cost of communication between the source and destinations in the network, measured by $\mathsf{bit}\times\mathsf{hop}$. As will be shown, even this simplest objective raises non-trivial challenges. To meet more complex overall goals, one should refer to the techniques developed in [4].

Consider the outgoing traffic of the source node 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 denoted by $d(u,v)$, which is measured as the minimum number of hops 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 memory-assisted compression gain. Accordingly, for each destination D, we define *effective walk*, denoted by $W_D=\{S,u_1,\ldots,D\}$, 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 that minimizes the $\mathsf{bit}\times\mathsf{hop}$ cost.

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

$$\mu_{D_i}=\arg\min_{\mu\in\boldsymbol{\mu}}\left\{\frac{d(S,\mu)}{g}+d(\mu,D_i)\right\}.$$

The total flow $\mathcal{F}_S$ of node S is then defined as

$$\mathcal{F}_S\triangleq\sum_{D_i\in\boldsymbol{D}_1}\left(\frac{f_{D_i}}{g}d(S,\mu_{D_i})+f_{D_i}d(\mu_{D_i},D_i)\right)+\sum_{D_j\in\boldsymbol{D}_2}f_{D_j}d(S,D_j). \tag{AA}$$

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

$$\hat{d}_D=\begin{cases}\frac{d(S,\mu_D)}{g}+d(\mu_D,D) & D\in\boldsymbol{D}_1\\ d(S,D) & D\in\boldsymbol{D}_2.\end{cases} \tag{2}$$

In short, the effective distance is in the presence of gain g obtained from memory-assisted compression. By definition, $\hat{d}_D\le d(S,D)\ \forall D$.

In a general network topology, the network compression gain (denoted by $\mathcal{G}$) as a function of memory-assisted compression gain g is defined as follows:

$$\mathcal{G}(g)\triangleq\frac{\sum_{S\in V}\mathcal{F}_S^0}{\sum_{S\in V}\mathcal{F}_S}=\frac{\sum_{S\in V}\sum_{D\in\boldsymbol{D}}d(S,D)}{\sum_{S\in V}\sum_{D\in\boldsymbol{D}}\hat{d}_D} \tag{3}$$

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

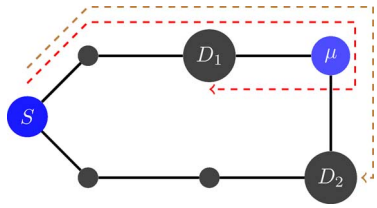


Fig. 7. Example of routing in networks featuring memory: introducing memory-enabled nodes can lead to changes in the effective shortest paths (shown by dashed lines). Here, $g = 3$.

scheme on top of what could be saved by universal compression (without memory) applied at the source and decoded at the destination. Alternatively, $\mathcal{G}(g)$ can be rewritten as

$$\mathcal{G}(g) = \frac{\sum_{S\in V}\sum_{D\in \boldsymbol{D}} d(S,D)El_n(X^n)}{\sum_{S\in V}\sum_{D\in \boldsymbol{D}} \left[d(S,\mu_D)El_{n|m}(X^n) + d(\mu_D,D)El_n(X^n)\right]}.$$

To demonstrate the challenges of the memory deployment problem and clarify the discussion, we consider one simple example network, which is presented in Fig. 7. Consider the destination node D_1, and let $g = 3$. The effective walks from the source to destinations are obviously the shortest paths when there is no memorization (Ucomp coding strategy defined in Section III). As shown in the figure, when the node μ is memory-enabled, the effective path to D_1 changes from the shortest path. Prior to memory deployment, the shortest path to D_1 was two hops long, while enabling μ with memorization completely changes the effective distance to D_1 to be $\hat{d}_{D_1} = \frac{3}{3} + 1 = 2$ as depicted in the figure. Note that in this example, we assumed no bandwidth constraint on the links. If D_1-μ link has a relatively small bandwidth, then the effective shortest path while minimizing the bit $\times$ hop might result in violating the bandwidth constraint on this link as the link would need to be used twice for passing one bit D_1. Taking such constraints into account would further complicate the problem and is not considered in this paper. The interested reader is referred to [4] for further details on such issues. Now, considering D_2, the cost of routing counter clockwise is 3, whereas it is $2 = \frac{3}{3} + 1$ by passing through the node μ clockwise. This example shows that introducing memory-enabled nodes can result in an effective shortest path that does not resemble the conventional shortest-path at all.

VI. Optimal Routing and Memory Placement Problem in Networks Compression

As demonstrated in the previous section, the deployment of memory-enabled nodes in the network gives rise to a number of questions and also brings some new challenges. We saw that the shortest path is not necessarily minimum cost route in networks with memory-enabled nodes, and hence, the well-known routing methods like Dijkstra's algorithm, in their original form are not optimal for networks with memory-enabled nodes.

We aim at answering two fundamental (and related) questions regarding memory-assisted network compression. In Section VI-A, we derive the best strategy to route packets between the source and destination nodes given the network topology, the location of the memories, and the gain g of memorization using a modification of Dijkstra's algorithm. In Section VI-B, we consider the problem of finding the best M nodes (out of N) to maximize the benefits of memory deployment.

A. Routing in Networks Featuring Memory

We consider an instance of network with a source node and fixed memory-enabled nodes and solve the routing problem in that instance of the network. Note that characterizing $\mathcal{G}$ involves computing both $\mathcal{F}^0$ and $\mathcal{F}$, which in turn requires finding the shortest paths and effective shortest paths between all pairs of nodes. The shortest path problem in a network without memory-enabled nodes is readily solved using Dijkstra's algorithm.

Finding the shortest path using the Bellman-Ford algorithm relies on the so-called principle of optimality: if a shortest path from u to v passes through a node w, then the portions of the path from u to w and from w to v are also shortest paths. It is notable that in the networks with memory-enabled nodes the modified principle of optimality reads as: given a shortest path from u to v, the portion of the path from v to w is still a shortest path whereas the portion from u to v is not necessarily a shortest path (This can also be seen in the example presented in Fig. 7). Hence, we will use this modified principle of optimality to find the effective shortest path using the well-known Bellman-Ford algorithm which is obtained by repeatedly applying 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, the more efficient Dijkstra's algorithm is used more widely in practice, most notably in *IS-IS* and *OSPF* (Open Shortest Path First) and hence we also visit the challenges of determining the effective shortest path in networks with memory-enabled nodes using the Dijkstra's algorithm.

The Dijkstra's algorithm solves the single-source shortest path for a network with positive edge costs, and hence, the bit $\times$ hop cost problem is a special case in which all the edge costs are equal to 1. Here, we present a modified version of Dijkstra's algorithm that determines the effective walk from all the nodes in a network to a destination D, in a network with a single memory-enabled node. 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. In a nutshell, to handle the memory node, we define a node-marking convention by defining a set $\mathcal{M}$ which contains the marked nodes. A node is marked if it is either a memory node, or it is a node through which a compressed flow is routed. The modified Dijkstra's algorithm starts with finding a node ν closest to node D. Then, it iteratively updates the effective distance of the nodes to D. The algorithm is summarized in Algorithm 1. The notation $\text{cost}(vD)$, used in Algorithm 1, is in fact the effective distance. At the beginning, the costs are initialized to $\text{cost}(vD) = \infty$ 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 determine the network compression gain $\mathcal{G}$.

Algorithm 1 Modified Dijkstra's Algorithm

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

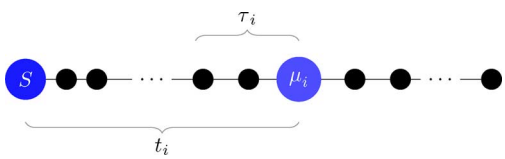


Fig. 8. The placement of memory units on a line network: the source node S is placed at one end of the line and the i-th memory is placed at t_i from the source and τ_i is its left-coverage.

B. Memory Placement in a Network Graph

The network compression gain depends on the number of memory-enabled nodes and also the locations they are deployed in the network. Since in practical scenarios only a select number of nodes have the storage and computational capability to function as a memory-enabled node, it is important to find the optimal location for such nodes. 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 the network compression gain $\mathcal{G}(g)$ is maximized. It can be shown that this problem can be reduced to the well-known k-median problem, and hence, it is an NP-hard problem on a general graph. In other words, no tractable optimal placement strategy exists for a given general graph.

In this section, we demonstrate the challenges of the memory deployment problem by considering the class of line networks for which we can obtain closed-form solutions. Solving the memory placement problem on this simple network topology will reveal why this problem is hard in general. 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 Fig. 8. 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 hop-distance t_i from the source. Without loss of generality, we also assume that $t_i < t_j$ for $i < j$, as shown in Fig. 8. We find t_i's such that total flow $\mathcal{F}$ is minimized (or equivalently, $\mathcal{G}(g)$ is maximized). A related problem of finding "en-route" memory deployment on line networks is studied in [48]. 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 [48] is $t_i = \frac{i}{M} \quad \forall \mu_i$.

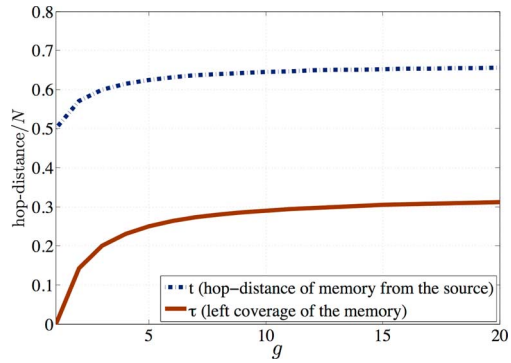


Fig. 9. Variations of t and τ vs. g for a line network.

On the other hand, the memory deployment problem for network compression on a line network is more challenging. The difficulty arises from the fact that each memory-enabled node can serve some of the destination nodes that are located at a smaller hop-distance from the source than the memory node itself. As shown in Fig. 8, 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 are affected when the memory-assisted compression gain g chanes.

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

$$t = \frac{2g}{3g+1}N + O(1), \tag{4}$$

$$\tau = \frac{g-1}{3g+1}N + O(1). \tag{5}$$

Proof: The total flow can be written as

$$\mathcal{F} = \int_0^{t-\tau} x\,\mathrm{d}x + \left(\frac{t(1-t)}{g} + \int_0^{1-t} x\,\mathrm{d}x\right) + \left(\frac{t\tau}{g} + \int_0^{\tau} x\,\mathrm{d}x\right). \tag{6}$$

The first term in (6) 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 (4) follows by taking the derivative of $\mathcal{F}$ and equating to zero, i.e., $\frac{\partial}{\partial t}\mathcal{F} = 0$ and $\frac{\partial}{\partial \tau}\mathcal{F} = 0$. ■

Fig. 9 shows the plot of t and τ versus g. as can be seen, as the gain g increases, the optimal place for the memory is on $\frac{2}{3}N$ distance from the source and the left coverage approaches $\frac{1}{3}N$, whereas for $g = 1$, the problem degenerates to that considered in [48], i.e., the left-coverage is zero and the optimal place for memory is at $\frac{1}{2}N$.

Lemma 2: The network-wide gain of placing a single memory ($M = 1$) on a line network of bit-hop length N as $N \to \infty$ converges to

$$\mathcal{G}(g) = \frac{(3g+1)^2}{3g^2+10g+3}.$$

Further, for $g \gg 1$ this converges to

$$\lim_{g\to\infty} \mathcal{G}(g) = 3.$$

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

According to Lemma 2, if only one memory is deployed in the network, even if the memory-assisted compression gain is infinite (i.e., effectively no bits need to be sent from the source node S to the memory node μ), the network compression gain is finite. This reveals that to achieve proper network-wide scaling, the number of memory-enabled nodes would also need to scale.

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

Theorem 3: Consider deployment of M memory-enabled nodes on a line where memory, where μ_i is placed at t_i and the left-coverage is denoted by τ_i. Then,

$$\begin{cases} t_i &= \frac{i}{M}N + O(1) \\ \tau_i &= \frac{g-1}{2gM}N + O(1). \end{cases} \tag{7}$$

Furthermore as $N \to \infty$, we have

$$\mathcal{G}(g) = \frac{2g^2 M}{2g(M+1) + g^2 + 1}$$

and for $g \gg 1$ we have

$$\lim_{g\to\infty} \mathcal{G} = 2M. \tag{8}$$

Proof: Similar to the proof of Lemma 1, we can write

$$\mathcal{F} = \sum_{m=1}^{M} \left[\frac{t_i}{g}\tau_i + \int_0^{\tau_i} x \, \mathrm{d}x + \frac{t_i}{g}(t_{m+1} - \tau_{m+1} - t_i) + \int_0^{t_{m+1}-\tau_{m+1}-t_i} x \, \mathrm{d}x \right].$$

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

$$\begin{cases} t_i &= \frac{t_{i+1}+t_{i-1}}{2} \\ \tau_i &= \frac{g-1}{2g}(t_i - t_{i-1}) \end{cases} \tag{9}$$

where (9) results in a tridiagonal matrix which in turn results in (7) for large M. Further, (8) follows from (7) and $\mathcal{F}_0 = 1/2$ for line networks. ■

Thus far, we showed that the memory deployment problem is non-trivial even on a line network and the optimal solution is indeed not very intuitive at the first glance. Furthermore, these problems would need to be modified when other constraints such as link bandwidths are present [4].

VII. Network Compression in Erdős-Rényi Random Network Graphs

In this section, we would like to analyze the minimum number of memory-enabled nodes required for the network-wide benefits of memory-assisted compression to be achievable in a large-scale network. In one extreme, if all of the network nodes are memory-enabled, almost automatically, the link gain would translate to the network-wide gain. On the other hand, if only a constant number of nodes participate, the performance improvement would not scale properly (as observed also for a line network in Section VI-B). This section aims at addressing what happens in between these extremes. To do so, we use Erdős-Rényi (ER) random network graphs. The main reason is that the symmetry in these network graphs yield to analytic closed-form solution that can be insightful when considering general network graphs.

A. Background on ER Random Graphs

Before we can state our main results, we need to cover the background on ER random graphs.

Definition 1: An ER random graph $G(N,p)$ is an undirected, unweighted graph on N vertices where any two vertices are connected with an edge with probability p.

Definition 2: Let $u, v \in G$ be any two vertices. The diameter of a connected graph is defined as $\max_{u,v} d(u,v)$. Further, the average distance of a connected graph is defined as $\mathbf{E}[d(u,v)]$.

The following properties hold for ER random graphs [49]:

1) $G(N,p)$ contains an average of $\binom{N}{2}p$ 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 $c \log N$ [50].
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

$$\bar{d} = (1 + o(1))\frac{\log N}{\log Np} \tag{10}$$

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

B. Main Result

To characterize the network compression gain, we consider connected $G(N,p)$, $p = \frac{c \log N}{N}$, with a single source node S and all other nodes as destinations (uniformly at random). Since the expected degree of all nodes in ER graph is equal and every vertex is chosen as a destination with equal probability, the optimal memory selection problem in this case disappears, and we select memories $\{\mu_i\}_{i=1}^{M}$ uniformly at random. Theorem 4 provides the scaling of $\mathcal{G}(g)$ with respect to M.

Theorem 4: Suppose M memory-enabled nodes are deployed on an ER random graph and let $\epsilon > 0$ be a positive real number. Then,

(a) *If $M = O\left(N^{\frac{1}{g}-\epsilon}\right)$, then $\mathcal{G}(g) \sim 1$.*[2]

(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, (10) results in

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

[2]Throughout this work, we have used the following asymptotic notations:
- $f(x) = o(g(x))$ iff $|f(x)| \le |g(x)|\epsilon$, $\forall\epsilon$,
- $f(x) = O(g(x))$ iff $|f(x)| \le |g(x)|k$, $\exists k$,
- $f(x) = \Omega(g(x))$ iff $|f(x)| \ge |g(x)|k$, $\exists k$, and
- $f(x) \sim g(x)$ iff $f(x)/g(x) \to 1$.

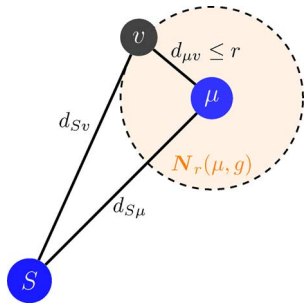


Fig. 10. Illustration of memory neighborhood.

Next, we need to find $\mathcal{F}$. For every memory μ we consider a neighbourhood $\boldsymbol{N}_r(\mu)$ as shown in Fig. 10. 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 node μ. Clearly, if $\frac{d(S,\mu)}{g} + r = d(S,v)$, the benefit provide by the memory for node v 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 $\boldsymbol{N}_r(\mu, g)$.

$$\boldsymbol{N}_r(\mu, g) = \left\{ v : \frac{d(S,\mu)}{g} + d(\mu, v) \le d(S, v) \right\}. \tag{11}$$

Since memory nodes 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 (11) and then using the result on the average distance in (10), we conclude

$$r \overset{a.s.}{=} (1 - 1/g) \left(\frac{\log N}{\log \log N} \right). \tag{12}$$

The following lemma, by Chung and Lu [49], 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 5 ([49]): 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 5 and (12), we deduce that

$$\begin{aligned} |\boldsymbol{N}_r(\mu_i, g)| &\overset{a.s.}{\le} (1 + 2\epsilon)(\log N)^{\left(1 - \frac{1}{g}\right)\left(\frac{\log N}{\log \log N}\right)} \\ &= (1 + 2\epsilon) N^{1 - 1/g}. \end{aligned} \tag{13}$$

Therefore, the total number of nodes that gain from the memory-enabled nodes is upper-bounded by

$$\sum_{i=1}^{M} |\boldsymbol{N}_r(\mu_i, g)| \le M(1 + 2\epsilon) N^{1 - 1/g}.$$

From (13), it is clear that the network compression gain vanishes if M is too small. The value $N^{1/g}$ is the threshold value for the network-wide gain. More accurately, if $M = O\left(N^{\frac{1}{g} - \epsilon}\right)$, then the memory-assisted compression gain would not result in any network-wide improvement. This is in contrast to the line network where with a single memory, we would obtain network-wide improvements.

Proof of the Main Result

Proof of Theorem 4(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

$$\mathcal{G}(g) \le \frac{N\bar{d}}{\frac{\bar{d}}{g} M |\boldsymbol{N}_r(\mu, g)| + \bar{d}(N - M|\boldsymbol{N}_r(\mu, g)|)} \tag{14}$$

$$\begin{aligned} &\overset{a.s.}{\le} \frac{N}{N - (1 - 1/g) M N^{\left(1 - \frac{1}{g}\right)}} \\ &= \frac{N}{N - (1 - 1/g) N^{1 - \epsilon}} \\ &\sim 1, \end{aligned} \tag{15}$$

where inequality in (14) follows from the double counting of the destination nodes that may reside in more than one neighborhood. Also, (15) follows from replacing (13) in (14). ■

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 invoke the following concentration inequality from [49] to establish the desired bound.

Proposition 6 ([49]): 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

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

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 7: 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

$$|\boldsymbol{N}_r(\mu, g)| \ge (1 - \epsilon)(Np)^r. \tag{16}$$

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,

$$\begin{aligned} \mathbf{E}\left[|\Gamma(V)|\right] &= \sum_{u \notin V} \mathbf{E}[X_u] \\ &= \sum_{u \notin V} \boldsymbol{P}[u \in \Gamma(V)] \\ &= \sum_{u \notin V} \left(1 - (1 - p)^{|V|}\right) \\ &\ge p|V|(N - |V|) \qquad (17) \\ &= (1 + o(1))Np|V| \qquad (18) \end{aligned}$$

where the inequality in (17) follows from

$$\begin{aligned} \boldsymbol{P}[u \in \Gamma(V)] &= 1 - (1 - p)^{|V|} \\ &\ge 1 - e^{-p|V|} \\ &\sim p|V|, \end{aligned}$$

and the second part holds because $\frac{|V|}{N} = o(1)$. Since, X_u's are non-negative independent random variables, by applying Proposition 6 with $\lambda = \sqrt{\alpha \mathbf{E}[|\Gamma(V)|]}$, with probability at least $1 - e^{-\alpha/2}$ we have

$$|\Gamma(V)| \geq \mathbf{E}\left[|\Gamma(V)|\right] - \sqrt{\alpha \mathbf{E}[|\Gamma(V)|]} \tag{19}$$
$$\geq (1-\epsilon)Np|V|. \tag{20}$$

By picking a single vertex and applying (19) inductively r times, and then adding up the number of adjacent nodes, we obtain (16). ∎

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:

$$r_\delta = (1 - 1/g - \delta)\left(\frac{\log N}{\log\log N}\right). \tag{21}$$

With this choice of r_δ, invoking Lemmas 5 and 7, 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

$$\begin{aligned}\mathbf{E}\left[\left|\bigcup_{i=1}^{M} \boldsymbol{N}_{r_\delta}(\mu_i, g)\right|\right] &= \sum_{u\in G} \boldsymbol{P}\left[u \in \cup_{i=1}^{M} \boldsymbol{N}_{r_\delta}(\mu_i, g)\right] \\ &= \sum_{u\in G}\left(1 - (1 - N^{-1/g-\delta})^M\right) \\ &\sim N\left(MN^{-1/g-\delta}\right) \\ &\sim N, \end{aligned} \tag{22}$$

where (22) holds by choosing $M = N^{1/g+\delta}$.

To show that the number of covered nodes is concentrated around its mean, we use Proposition 6 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

$$\left|\bigcup_{i=1}^{M} \boldsymbol{N}_{r_\delta}(\mu_i, g)\right| \geq \mathbf{E}\left[|\cup \boldsymbol{N}_{r_\delta}(\mu_i, g)|\right] - \lambda \tag{23}$$
$$\geq (1 - o(1))N. \tag{24}$$

Hence, the memory-enabled nodes 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(b): Using (22), we can bound the network-wide gain of the memory from below. We have

$$\mathcal{G}(g) \overset{a.s.}{=} \frac{N\bar{d}}{(\bar{d}/g + r_\delta)N} \tag{25}$$
$$= \frac{1}{1/g + (1 - 1/g - \delta)} \tag{26}$$
$$= \frac{1}{1-\delta}, \tag{27}$$

where (25) holds because the distance of the nodes from memory is r_δ, asymptotically almost surely. Observe that as the number of memories becomes close to N, i.e., $\delta \to \left(1 - \frac{1}{g}\right)$, the gain $\mathcal{G} \to g$. ∎

VIII. Network Compression in Internet-Like Power-Law Random Network Graphs

In the previous section, using Erdős-Rényi random graphs, we demonstrated that if the number of memory-enabled nodes M increases like $N^{\frac{1}{g}}$, where N is the total number of nodes, and g is the memory-assisted compression gain, the network-wide benefits start to shine. Note that for any $g > 1$, as $N \to \infty$ we have

$$\lim_{N\to\infty} \frac{N^{\frac{1}{g}}}{N} = 0.$$

Therefore, even using an asymptotically vanishing number of memory-enabled nodes, we can obtain network-wide improvements. On the other hand, we expect even better network-wide scaling when the network has more structure than the Erdős-Rényi random graphs.

The focus of the study of this section is to extend network compression to random power-law graph (RPLG) model. The power-law graphs are particularly of interest because they are one of the useful mathematical abstractions 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 [17]–[21]. 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. 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.

A. Random Power-Law Graph Model

A random 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 [49]. 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 3 (Definition of $G(\beta)$): Consider the sequence of the expected degrees $\boldsymbol{w} = \{w_1, w_2, \ldots, w_N\}$, and let $\rho = 1/\sum w_i$. For every two vertices v_i and v_j, the edge v_iv_j exists with probability $p_{ij} = w_iw_j\rho$, independent of other edges. If

$$w_i = ci^{-\frac{1}{\beta-1}} \text{ for } i_0 \leq i \leq N + i_0 \tag{28}$$

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 [49], 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-enabled nodes.

B. Memory Deployment in Random Power-Law Graphs

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 9 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 3, 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 8 below is our main result regarding the size of the core:

Theorem 8: 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

$$l^{3-\beta} - \frac{1}{\bar{w}\gamma} = 0 \tag{29}$$

and the constant γ is equal to $\left(1 - \frac{1}{\beta-1}\right)^2 \frac{\beta-1}{3-\beta}$. 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 8 follows from the lemmas below.

Lemma 9: 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 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 9, 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 10 ([49]): A random graph $G(\beta)$ with the expected degrees $\boldsymbol{w}$, almost surely has no giant components if

$$\frac{\sum_i w_i^2}{\sum_i w_i} < 1. \tag{30}$$

Lemma 11: 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 8: Consider a $G(\beta)$ with the set of lowest degree nodes U_l, all having expected degrees in the interval $(w_{\min}, lw_{\min})$. According to Lemma 10, to ensure that the induced subgraph G_{U_l} does not have a giant component, we should have

$$\sum_{v \in U_l} {w_v'}^2 / \sum_{v \in U_l} w_v' < 1,$$

$$\text{where} \qquad w_v' = w_v \frac{\sum_{v \in U_l} w_v}{N\bar{w}}$$

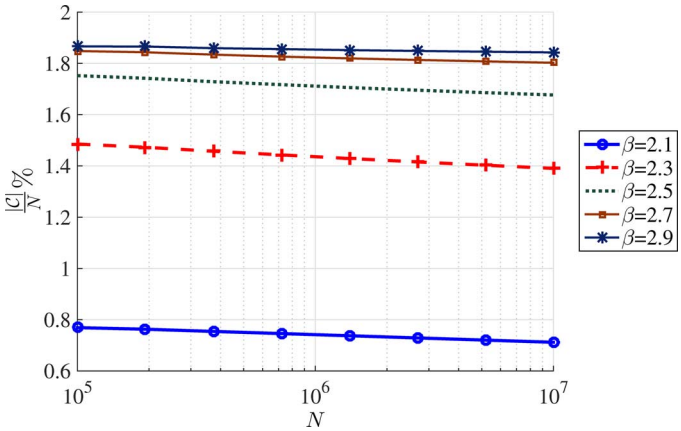


Fig. 11. The scaling of the core size $\frac{|\mathcal{C}|}{N} \times 100$ versus N for different β's.

as in Lemma 11. To find w', we should first obtain $\sum_{v \in U_l} w_v$. According to [49], we have

$$\sum_{v \in U_l} w_v \approx N\bar{w}(1 - l^{2-\beta}),$$
$$\sum_{v \in U_l} {w_v}^2 \approx N\bar{w}^2 \left(1 - \frac{1}{\beta - 1}\right)^2 \frac{\beta - 1}{3 - \beta} l^{3-\beta}. \quad (31)$$

From (31), we conclude that $w'_v = (1 - l^{2-\beta}) w_v$, for all $v \in U_l$. Thus we have,

$$\sum_{v \in U_l} w'_v \approx N\bar{w}(1 - l^{2-\beta})^2. \quad (32)$$

Similarly,

$$\sum_{v \in U_l} {w'_v}^2 \approx N\bar{w}^2 \gamma l^{3-\beta} (1 - l^{2-\beta})^2. \quad (33)$$

Combining (32) and (33), we obtain the relation in (29) 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 (28). ■

Theorem 8 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.

In Fig. 11 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.

C. 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 [51], with which we generated random power-law graph instances with number of vertices between 1,000 and 5,000, and $2 < \beta < 3$. The result are averaged over 5 instances of generated RPLGs. In our simulations, we report results for various core sizes (which is the number of memory-enabled nodes).

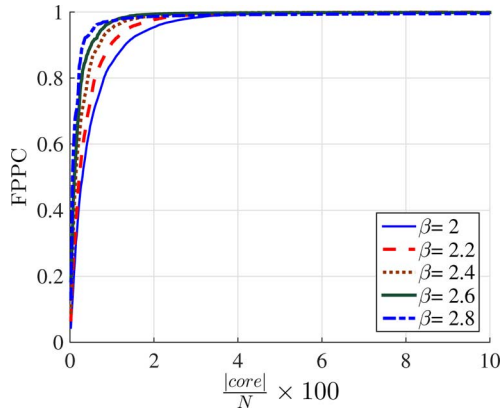


Fig. 12. The fraction of the paths passing through the core (FPPC) vs. the core size, for RPLG of size $N = 5,000$.

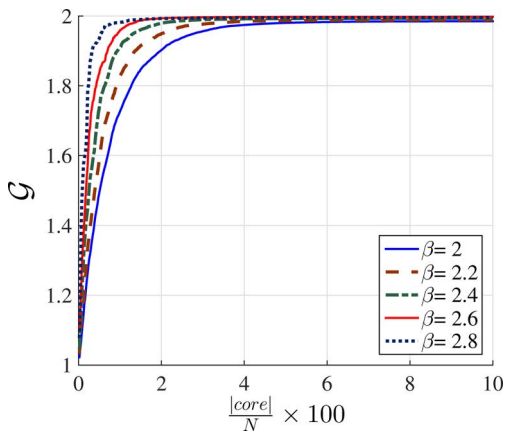
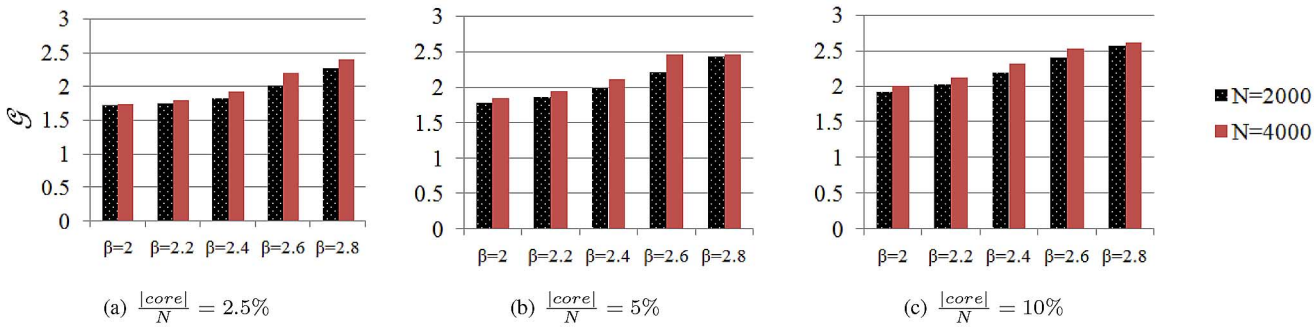


Fig. 13. Illustration of the network-wide gain when simple Dijkstra's routing algorithm is used for $g \to \infty$. Note that $\mathcal{G}$ is capped to two despite g growing large.

We first verify our assumption that a tiny fraction of the highest degree nodes observes most of the traffic in the network. Fig. 12 shows the Fraction of the Paths Passing through the Core (FPPC) for different core sizes and β's. As we expected, more that 90% of the shortest paths in the graph involve less than 2% of the highest degree nodes to route the flow. Although our theoretical result in Theorem 8 is asymptotic in N, Fig. 12 suggests that the asymptotic result holds for $N = 1,000$ already. 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 compression gain, we considered two RPLGs with sizes $N = 2,000$ and $N = 4,000$. 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 1 kB. This assumption is in accordance with the maximum transmission unit (MTU) of 1,500 bytes allowed by Ethernet at the network layer. From our results in Section IV-A, we expect that a memory-assisted compression gain of $g \approx 2.5$ is achievable for real traffic traces. Hence, we use $g = 3$ in our simulations. Please note that in the simulations of this section, the impact of the source statistics and compression is only considered through g.



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

Fig. 14. Network-wide gain of memory assisted compression $\mathcal{G}$ for different core sizes and power-law parameter β, for $g = 3$. (a) $\frac{|core|}{N} = 2.5\%$. (b) $\frac{|core|}{N} = 5\%$. (c) $\frac{|core|}{N} = 10\%$.

To verify the results of routing with memory-enabled nodes in Section VI, we conducted 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 9 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 Fig. 13), which is not desirable. Fig. 14 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 conventional end-to-end compression technique in Ucomp). Further, this feature only requires extra computation at the memory units when compared to Ucomp. The complexity of these operations scale linearly 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, network compression has the potential to reduce the traffic by a factor of 2.

IX. Conclusion

In this paper, we employed the concept of memory-assisted compression and introduced its implication in reducing the amount of traffic in networks. The basic idea is to allow some intermediate nodes in the network to be capable of memorization and compression. The memory-enabled nodes observe the traffic traversing the network and form a model for the information source. Then, using the side information from this source model, a better universal compression of the flow is achieved on the network flow. We investigated, from an information-theoretic point of view, the network flow compression by utilizing memory-enabled nodes in the network and solved the routing problem for networks with memory units. We also considered Erdős-Rényi random graphs and Internet-like power-law graphs to develop theoretical results on the number of memory-enabled nodes needed to obtain the network-wide benefits. Finally, our simulations demonstrated that by enabling memorization on less than 2.5% of the nodes in an Internet-like random power law graph, we can expect almost all of the network-wide benefits of memorization providing two-fold gain over conventional end-to-end universal compression.

Acknowledgment

The authors are grateful to Georgia Tech Networks and Mobile Computing (GNAN) Research Group for providing the traces of wireless network users used in the simulations of this paper. The authors also acknowledge very helpful discussions with R. Sivakumar about the benefits of combining de-duplication and memory-assisted compression. The authors are also grateful to the anonymous reviewers for their detailed comments that helped to improve the presentation of the paper.

References

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

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

[3] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee, "Redundancy in network traffic: Findings and implications," in *SIGMETRICS '09: Proc. 11th Int. Joint Conf. Measurement and Modeling of Computer Systems*, New York, NY, USA, 2009, pp. 37–48.

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

[5] Z. Zhuang, C.-L. Tsao, and R. Sivakumar, "Curing the Amnesia: Network Memory for the Internet," Tech. Report Georgia Inst. Technol., Atlanta, GA, USA, 2009 [Online]. Available: http://www.ece.gatech.edu/research/GNAN/archive/tr-nm.pdf

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

[7] S. C. Rhea, K. Liang, and E. Brewer, "Value-based web caching," in *Proc. 12th Int. Conf. World Wide Web, WWW '03*, New York, NY, USA, 2003, pp. 619–628.

[8] M. C. Chan and T. Y. C. Woo, "Cache-based compaction: A new technique for optimizing web transfer," in *Proc. IEEE INFOCOM '99*, New York, NY, USA, 1999.

[9] M. Sardari, A. Beirami, and F. Fekri, "On the network-wide gain of memory-assisted source coding," in *Proc. 2011 IEEE Information Theory Workshop (ITW)*, Oct. 2011, pp. 476–480.
[10] M. Sardari, A. Beirami, and F. Fekri, "Memory-assisted universal compression of network flows," in *Proc. IEEE INFOCOM*, Orlando, FL, USA, Mar. 2012, pp. 91–99.
[11] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inf. Theory*, vol. 23, no. 3, pp. 337–343, May 1977.
[12] F. Willems, Y. Shtarkov, and T. Tjalkens, "The context-tree weighting method: Basic properties," *IEEE Trans. Inf. Theory*, vol. 41, no. 3, pp. 653–664, May 1995.
[13] D. Baron and Y. Bresler, "An O(N) semipredictive universal encoder via the BWT," *IEEE Trans. Inf. Theory*, vol. 50, no. 5, pp. 928–937, May 2004.
[14] N. Merhav and M. Feder, "A strong version of the redundancy-capacity theorem of universal coding," *IEEE Trans. Inf. Theory*, vol. 41, no. 3, pp. 714–722, May 1995.
[15] A. Beirami and F. Fekri, "Results on the redundancy of universal compression for finite-length sequences," in *Proc. IEEE Int. Symp. Information Theory (ISIT)*, 2011, pp. 1504–1508.
[16] A. Beirami, M. Sardari, and F. Fekri, "Results on the fundamental gain of memory-assisted universal source coding," in *Proc. IEEE Int. Symp. Information Theory (ISIT '2012)*, Jul. 2012, pp. 1092–1096.
[17] R. Albert, H. Jeong, and A.-L. Barabasi, "Internet: Diameter of the world-wide web," *Nature*, vol. 401, pp. 130–131, 1999.
[18] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the Internet topology," in *ACM SIGCOMM*, 1999, pp. 251–262.
[19] A. Broder *et al.*, "Graph structure in the web," in *Proc. WWW9 Conf.*, 2000, pp. 309–320.
[20] A. Vázquez, R. Pastor-Satorras, and A. Vespignani, "Large-scale topological and dynamical properties of the Internet," *Phys. Rev. E*, vol. 65, no. 6, p. 066130, Jun. 2002.
[21] M. Boguná, F. Papadopoulos, and D. Krioukov, "Sustaining the Internet with hyperbolic mapping," *Nature Commun.*, vol. 1, p. 62, Sep. 2010.
[22] M. Mahoney, "Adaptive weighing of context models for lossless data compression," Tech. Rep. Florida Inst. Technol., Melbourne, FL, USA, 2005.
[23] P. Erdős and A. Rényi, "On random graphs. I," *Publicationes Mathematicae*, pp. 290–297, 1959.
[24] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," in *ACM SIGCOMM*, 2007, pp. 181–192.
[25] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. Briggs, and R. Braynard, "Networking named content," in *Proc. 5th ACM CoNEXT*, 2009, pp. 1–12.
[26] Z. Zhuang, T.-Y. Chang, R. Sivakumar, and A. Velayutham, "Application-aware acceleration for wireless data networks: Design elements and prototype implementation," *IEEE Trans. Mobile Comput.*, vol. 8, no. 9, pp. 1280–1295, Sep. 2009.
[27] J. Bentley and D. McIlroy, "Data compression using long common strings," in *Proc. Data Compression Conf., DCC '99*, 1999, pp. 287–295.
[28] R. M. Karp and M. O. Rabin, "Efficient randomized pattern-matching algorithms," *IBM J. Res. Devel.*, pp. 249–260, 1987.
[29] U. Manber, "Finding similar files in a large file system," in *Proc. USENIX Winter Tech. Conf.*, 1994.
[30] D. Slepian and J. K. Wolf, "Noiseless coding of correlated information sources," *IEEE Trans. Inf. Theory*, vol. 19, pp. 471–480, 1973.
[31] M. Sartipi and F. Fekri, "Distributed source coding using short to moderate length rate-compatible LDPC codes: The entire Slepian-Wolf rate region," *IEEE Trans. Commun.*, vol. 56, no. 3, pp. 400–411, Mar. 2008.
[32] G. G. Langdon Jr., "An introduction to arithmetic coding," *IBM J. Res. Develop.*, vol. 28, no. 2, pp. 135–149, Mar. 1984.
[33] F. Willems, "The context-tree weighting method: Extensions," *IEEE Trans. Inf. Theory*, vol. 44, no. 2, pp. 792–798, Mar. 1998.
[34] D. Shkarin, "PPM: One step to practicality," in *Data Compression Conf.*, 2002, vol. 12.
[35] S. Bunton, "On-line stochastic processes in data compression," Ph.D. dissertation, Univ. Washington, Seattle, WA, USA, 1996.
[36] D. Salomon, *Data Compression: The Complete Reference*. New York, NY, USA: Springer, 2007.
[37] C. Mattern, "Mixing strategies in data compression," in *2012 Data Compression Conf. (DCC)*, Snowbird, UT, USA, 2012.
[38] D. P. Bertsekas, *Nonlinear Programming*. Belmont, MA, USA: Athena Scientific, 1999.
[39] J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Inf. Theory*, vol. 24, no. 5, pp. 530–536, Sep. 1978.
[40] R. Lenhardt and J. Alakuijala, "Gipfeli – high speed compression algorithm," in *2012 Data Compression Conf. (DCC)*, Snowbird, UT, USA, 2012, pp. 109–118.
[41] S. Gunderson, Snappy [Online]. Available: http://code.google.com/p/snappy/
[42] A. Hidayat, Fastlz [Online]. Available: http://www.fastlz.org
[43] L. M. Reinhold, Quicklz [Online]. Available: http://www.quicklz.com
[44] N. Krishnan and D. Baron, "A universal parallel two-pass MDL context tree compression algorithm," *IEEE J. Sel. Topics Signal Process.*, vol. 9, no. 4, pp. 1–8, Jun. 2015.
[45] Wireshark Packet Analyser. [Online]. Available: http://www.wireshark.org/
[46] A. Beirami and F. Fekri, "A novel correlation model for universal compression of distributed parametric sources," in *52nd Annu. Allerton Conf. Communication, Control, and Comput.*, 2014.
[47] L. Huang, A. Beirami, M. Sardari, F. Fekri, B. Liu, and L. Gui, "Packet-level clustering for memory-assisted compression of network packets," in *2014 Int. Conf. Wireless Communications and Signal Processing (WCSP 2014)*, 2014.
[48] P. Krishnan, D. Raz, and Y. Shavitt, "The cache location problem," *IEEE/ACM Trans. Netw.*, vol. 8, pp. 568–582, 2000.
[49] F. Chung and L. Lu, *Complex Graphs and Networks*. Providence, RI, USA: American Mathematical Society, 2006.
[50] N. Alon and J. Spencer, *The Probabilistic Method*, 3rd ed. New York, NY, USA: Wiley, 2008.
[51] A. Brady and L. Cowen, "Compact routing on power law graphs with additive stretch," in *ALENEX*, 2006.

Ahmad Beirami (S'07) received the B.Sc. degree in electrical engineering from Sharif University of Technology, Tehran, Iran, in 2007 and the M.Sc. and Ph.D. degrees in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 2011 and 2014, respectively.

He is currently a Postdoctoral Associate jointly affiliated with the information initiative at Duke (iiD) and the Research Laboratory of Electronics (RLE) at MIT. His research interests broadly include information theory, cyber security, machine learning, statistics, and networks.

Dr. Beirami is the coauthor of a paper that received a Best Student Paper nomination in IEEE Midwest Symposium on Circuits and Systems (2008). His Ph.D. work received the Center for Signal and Information Processing Outstanding Research Award (2014), the 2013–2014 School of ECE Graduate Research Excellence Award, and the 2015 Sigma Xi Best Ph.D. Thesis Award, all from Georgia Institute of Technology.

Mohsen Sardari received the B.Sc. degree in electrical engineering from Sharif University of Technology, Tehran, Iran, in 2007. He received the M.S.E.C.E. and Ph.D. degrees from the School of Electrical and Computer Engineering (ECE), Georgia Institute of Technology, Atlanta, GA, in 2010 and 2013, respectively.

He is currently a Data Scientist with Electronic Arts, Inc., Redwood City, CA, USA. His research interests broadly include information theory, signal processing, large-scale data analytics, and machine learning.

Faramarz Fekri (M'00–SM'03) received the Ph.D. degree from the Georgia Institute of Technology, Atlanta, GA, USA, in 2000.

Since 2000, he has been with the faculty of the School of Electrical and Computer Engineering at the Georgia Institute of Technology where he currently holds a Professor position. His current research interests are in the area of communications and signal processing, in particular source and channel coding, information theory in biology, statistical inference in large data, information processing for wireless and sensor networks, and communication security.

Prof. Fekri received the National Science Foundation CAREER Award (2001), Southern Center for Electrical Engineering Education (SCEEE) Young Faculty Development Award (2003), and Outstanding Young Faculty Award of the School of ECE (2006). He serves on the Technical Program Committees of several IEEE conferences. In the past, he was on the editorial board of the IEEE TRANSACTIONS ON COMMUNICATIONS, and the Elsevier *Journal on PHYCOM*.