Content-Aware Network Data Compression Using Joint Memorization and Clustering

Mohsen Sardari,† Ahmad Beirami,† Jun Zou, Faramarz Fekri
School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332
Email:{mohsen.sardari, beirami, junzou, fekri}@ece.gatech.edu

***Abstract*—Recent studies have shown the existence of considerable amount of packet-level redundancy in the network flows. Since application-layer solutions cannot capture the packet-level redundancy, development of new content-aware approaches capable of redundancy elimination at the packet and sub-packet levels is necessary. These requirements motivate the redundancy elimination of packets from an information-theoretic point of view. For efficient compression of packets, a new framework called memory-assisted universal compression has been proposed. This framework is based on learning the statistics of the source generating the packets at some intermediate nodes and then leveraging these statistics to effectively compress a new packet. This paper investigates both theoretically and experimentally the memory-assisted compression of network packets. Clearly, a simple source cannot model the data traffic. Hence, we consider traffic from a complex source that is consisted of a mixture of simple information sources for our analytic study. We develop a practical code for memory-assisted compression and combine it with a proposed hierarchical clustering to better utilize the memory. Finally, we validate our results via simulation on real traffic traces. Memory-assisted compression combined with hierarchical clustering method results in compression of packets close to the fundamental limit. As a result, we report a factor of two improvement over traditional end-to-end compression.**

I. Introduction

To cope with the ever increasing amount of data transmitted in the data networks we should either increase the backbone capacity or find ways to improve link efficiency, i.e., decrease the amount of transmitted data. The underlying fabric of the networks perform very little, if any, memorization. The only form of memorization commonly performed in the network is application-layer caching used by solutions such as web-caches, content-distribution networks (CDNs), and peer-to-peer (P2P) applications. Recently, many researchers have rethought this aspect of the networks by equipping some nodes in the network with memorization capability in order to perform better redundancy elimination via deduplication (cf. [1], [2]).

However, there is a lot to be gained beyond the simple deduplication if we exploit the statistical redundancies within a packet as well as significant dependencies that exist across packets. These statistical redundancies can potentially be suppressed using variable length compression techniques. However, for an IP packet with a length only approximately 1500 bytes, the traditional compression techniques [3], [4] are inefficient in capturing the redundancy in data as compression performance primarily depends on the sequence length (cf. [5] and the references therein). In other words, there is a significant penalty with respect to what is fundamentally achievable when we attempt to universally compress a finite-length packet [5]. Further, since each packet may be destined to a different user, these packets cannot be effectively compressed with traditional end-to-end compression techniques which cannot leverage the cross packet dependencies. Therefore, it is very desirable to encode and compress each individual packet more efficiently by utilizing the dependency and side-information provided from the rest of the packets (while we still deliver each packet separately). In short, an investigation for a protocol-independent and content-aware network packet compression scheme suitable for removing redundancy within each packet and the dependency across multiple packets is in order.

A new framework for compression of small sequences, called *memory-assisted compression*, has been recently developed and its potential application in network compression has been introduced [6]–[8]. In memory-assisted framework, compression of a sequence is performed using a memory of the previously seen sequences. Consequently, every sequence can be compressed far better compared to the case that the sequence is compressed by its own without considering the memory.

One major challenge to analytically study the performance of memory-assisted compression on real traffic traces is how to model the content generator. Clearly, a single stationary source[1] does not fully model a real content generator, such as a web-server, as it may contain text, html code, images and video. Instead, a better model is to view every content generator as a compound (mixture) of several stationary information sources whose true statistical models are not available. As shown in Fig. 1, a compound source can be thought of as a set of $\mathcal{K}$ stationary sources $S_1, \ldots, S_{\mathcal{K}}$ each having its own statistical model. We assume that, metaphorically, the server is communicating through a memory element M_1 with the rest of the network. In reality, M_1 could be physically attached to the server. Corresponding to M_1 there is another memory element M_2 in the network that serves the clients denoted by C.

Memory element M_2 could be a router or a gateway in the network that is equipped with memory. The two memory elements M_1 and M_2 have observed the past communication packets from the server to the various clients and obtained a common shared memory. We wish to characterize the memory-assisted compression benefit provided in the M_1-M_2 link as a function of the memory size shared between M_1 and M_2.

This material is based upon work supported partially by the National Science Foundation under Grant No. CNS-1017234.

† Both authors contributed equally to this paper.

[1]The statistics of a stationary information source remain unchanged in time.

978-1-4673-5946-7/13/$31.00 ©2013 IEEE

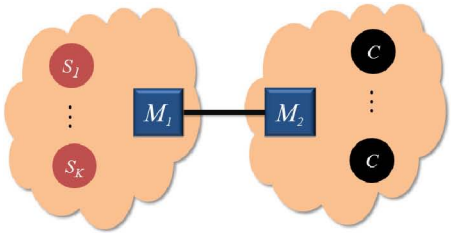

Fig. 1. The basic memory-assisted compression scenario between two memory elements M_1 and M_2 in the network. The compound source (i.e., the content server) is shown as a set of multiple simple sources $S_1, \ldots, S_{\mathcal{K}}$ on the left.

This setup occurs in many applications, such as communication in the Internet between two major hubs, communication in enterprise networks, etc.

This paper is composed of two parts. In the first part, we consider a set of sequences generated from a compound source and we investigate the theoretical results on the performance of memorization and clustering for compression of a newly generated sequence from the source. Specifically, we first find that, without clustering of the sequences in the memory, memorization of a compound source is not always beneficial and it may even degrade the compression performance. Then, we closely examine the theoretical gain of the joint memorization and clustering for the compression of the compound source.

In the second part, we develop and study a content-aware clustering algorithm which aims at utilizing the data in the memory to better compress a new sequence from the compound source. The main idea of the clustering is to group packets in memory that can be compressed well together. Packets in the same cluster would share similar statistical properties hence improving the compression performance. A newly generated packet by the compound source is first classified into one of the clusters and then the set of packets in the selected cluster is used as the context memory for the compression of the packet.

II. Related Work

The topic of redundancy elimination in network data traffic has recently received a lot of attention [2], [9]–[12]. This series of work is motivated by the observation that there exist a considerable amount of duplicate strings within the network packets. Since, by the design limitation, application and object-level caching mechanisms cannot capture such duplicates within packets, the redundancy elimination techniques operate below the application layer at the packet level. Therefore, they can suppress any duplicate strings of bytes that appear on a single link or they can be deployed on a wider scale across multiple network routers enabling an IP-layer protocol-independent redundancy elimination service.

In [6], we introduced the concept of network compression (via network memory) and investigated its trade-offs in random Erdős-Renyi graphs. In [6], we assumed a value for achievable gain of memory-assisted compression on a single link (called memorization gain g) and investigated the benefits of network compression via memory. In [7], we obtained typical values for g for a simple stationary information source, such as the script files of a web-server and then extended the results of [6] and evaluated network-wide gain of network compression in an Internet-like random power-law graph as function of

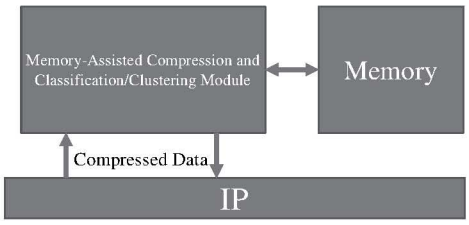

Fig. 2. Network Compression architecture which includes the classification/clustering module.

link compression gain. However, the typical values that the gain g can assume on real-world data traces from complex sources in the network remained unexplored. In this paper, we both characterize the performance improvement achieved via memory-assisted compression and devise practical algorithms to achieve such improvement on every packet from real-world traffic, on the fly with low complexity.

III. Setup

The core to our approach for network data compression is network memory. In a nutshell, memory enabled nodes in network can learn the data statistics of traffic which can then be used (as side information) in compression toward reducing the cost of describing the information source generating the data in compression. Therefore, network compression via memory can be a new layer 3.5 in the network, as shown in Fig. 2. The network compression layer may be present at any entity such as the server or the router. Further, it should understand the transport layer and network layer semantics. The sender-side operations and the receiver-side operations are different in the sense that the sender encodes the data and the receiver decodes the data. However, the operations regarding the maintenance of the memory should be the same on both sides to guarantee that the side information needed to remove the redundancy in the network traffic is the same at both ends. The architecture in Fig. 2 also includes a clustering/classification module, the introduction of which is motivated by the following.

A fundamental question is whether or not storing the previous packets offers any benefit for the compression of a new data packet. To answer this question, we did experiments with two sets of data: Dataset 1, which is gathered from a mixture of data packets from CNN and Apple websites, and Dataset 2, which is merely *script* files gathered from CNN website. First, we applied the memory-assisted compression techniques developed in [7] for a stationary source on dataset 1, to which hereafter we refer as *naive memory-assisted compression*. This naive scheme does not take into account that the packets in memory are from the compound source. Our initial assessments demonstrated that, on the average, the naive memory-assisted compression does not provide any compression benefit.

The results from this experiment in Table I show that the compression without memory achieves an expected compression rate of about 5.41 bits/byte as compared with 8 bits/byte required for the representation of each data byte when no compression is in effect. Further, the naive memory-assisted compression achieves a compression rate of around 5.32 bits/byte and offers little compression benefit. On the other hand, when

TABLE I
THE UNIVERSAL COMPRESSION RATE ON DATASET 1 WITH DIFFERENT CODING STRATEGIES: MEMORY-ASSISTED CTW AND LZW ALGORITHMS.

Compression Scheme	**compression rate** (bits/byte)	
	CTW	LZW
No Compression	8	8
No Memorization (Ucomp)	5.41	6.27
Naive Memorization (UcompM)	5.32	5.01
This Work (UcompH)	2.80	3.63

memory-assisted compression is applied only on the script files (i.e., dataset 2) significant compression enhancement was reported in [7]. Note that the dataset 2 may be considered as a single stationary source; the statistics between the script files does not undergo significant variation. Hence, this discrepancy in the compression benefit from memorization between these two datasets can only be attributed to the compound nature of the memorized packets in dataset 1. Therefore, to achieve compression benefit, we may have to properly partition dataset 1 before extracting the side information.

This observation motivated us to further study the joint memorization and *clustering* for a compound source, where the memorized packets are partitioned into several clusters for better compression performance. The result is reported in Table I as UcompH. In the rest of this paper, we first theoretically study the clustering problem. Then, we develop an efficient clustering algorithm for compression.

IV. THEORETICAL ANALYSIS

For analytical study, our model of content generator for the traffic is a mixture of several stationary (parametric) sources which is expected to model the complex nature of the content generator. Let $\mathcal{A}$ be a finite alphabet and let the parametric source be defined using a d-dimensional parameter vector $\theta = (\theta_1, ..., \theta_d)$, where d denotes the number of the source parameters. For example, if the alphabet size is $||\mathcal{A}|| = 256$, for a first-order Markov source the number of source parameters is 256×255 which is equal to the number of independent transition probabilities. Denote μ_θ as the probability measure defined by the parameter vector θ on sequences (packets) of length n. We also use the notation μ_θ to refer to the parametric source itself. We assume that the d parameters are unknown. We use the notation $x^n = (x_1, ..., x_n) \in \mathcal{A}^n$ to present a packet of length n from the alphabet $\mathcal{A}$, i.e., a packet contains n bytes.

For our analysis, we assume that, in Fig. 1, both the encoder (at M_1) and the decoder (at M_2) have access to a common memory of the previous T packets (each of size n) from the compound source. In other words, the memory size is $m = nT$. Further, denote $\mathbf{y}$ as the concatenation of the previous T packets shared between M_1 and M_2. Consider the communication scenario in Fig. 1. The presence of the shared memory $\mathbf{y}$ at M_1 and M_2 can be used by the encoder at M_1 to compress (via memory-assisted source coding) the packet x^n which is requested by client C. The compression can reduce the transmission cost on the M_1-M_2 link while being transparent to the client, i.e., M_2 decodes the memory-assisted code, then applies conventional compression on x^n and forwards to C.

To investigate whether or not memorization provides compression benefit for the compound source, we compare the following three schemes:

- Ucomp (Universal compression): a simple compression is applied on the packet x^n without considering $\mathbf{y}$.
- UcompM (Universal compression with naive context memorization): the encoder at M_1 and the decoder at M_2 both have access to packets $\mathbf{y}$ from the compound source; they use $\mathbf{y}$ for the compression of x^n, but without considering which source has generated the packets.
- UcompCM (Universal compression with source-defined clustering of the memory): assumes that the memory $\mathbf{y}$ is shared between M_1 and M_2. Further, both M_1 and M_2 know the index of the source (in the compound source) that has generated the memorized packets.

Next, we provide qualitative discussion on the performance of the different packet coding strategies introduced above by the analysis of the average minimax redundancy. The formal analysis and the proof sketches are omitted. The performance of traditional universal compression has been extensively studied in the literature (cf. [5] and the references therein). It is concluded that the performance of universal compression on finite-length packets, with size similar to IP packets, is fundamentally limited by the inevitable compression overhead (code redundancy) imposed by universal compression [5].

In the rest of this section, we present theoretical results on the performance of UcompM and UcompCM coding strategies. The case $\mathcal{K} = 1$ is the special case where all of the packets are from a single stationary source model, which also theoretically quantifies our previous results in [7].

Case $\mathcal{K} = 1$: In this case, there is no distinction between UcompM and UcompCM. It can be shown that when the memory size is large enough, i.e., sufficient number of packets from previous communication have been stored, the compression overhead becomes negligibly small [8].

UcompM: Case $\mathcal{K} \geq 2$: As stated in the problem setup, the packets in the memory are from a compound source. Results in Table I, suggest that naive memorization does not offer much improvement in compression performance. This observation is analytically justified in [8], where we show that the naive memorization of the previous packets using UcompM without regard to which source parameter has indeed generated the packet would not suffice to achieve the memorization gain. In fact, the redundancy of UcompM is worse than the redundancy of Ucomp, for large n. Therefore, the naive memorization of the context by node M_2 in Fig. 1 from the previous communications not only does not improve the compression performance but also asymptotically makes it worse. Next, we look at the compression performance in an ideal case where we know the source of each packet.

UcompCM: Case $\mathcal{K} \geq 2$: Thus far, we learned that the naive memorization in the memory element is not beneficial when a compound source is present. This necessitates to first appropriately *cluster* the packets in the memory. Then, based on the criterion as to which cluster the new packet x^n belongs to, we utilize the corresponding memorized context for the compression. In source-defined clustering, we assume both M_1 and M_2 (in Fig. 1) exactly know the index $i \in [\mathcal{K}]$ and hence all the packets that belong to the same source $\theta^{(i)}$ in the compound

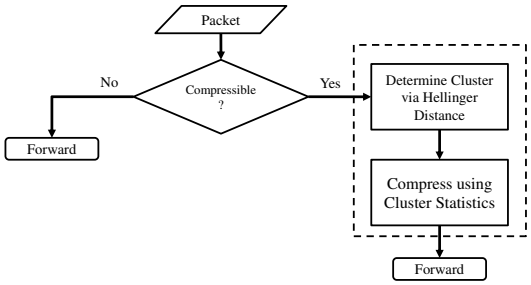

Fig. 3. Network packet compression flowchart. The modules in the dashed box are the components of the K-means clustering using Hellinger distance.

source are assigned to the same cluster (for all $i \in [\mathcal{K}]$). In [8], we theoretically justified that if sufficient memory of the past is present at the memory element, the compression overhead (redundancy) may be eliminated via joint memorization and clustering, hence improved compression performance. This result motivates the investigation of memorization and clustering approach in real-world scenarios. In Sec. V, we will further relax the source-defined clustering assumptions and study the impact of clustering in practice.

V. Hierarchical Content-Aware Clustering

In this section, we try to answer the main question in the memory-assisted compression setup we introduced: "How do we utilize the available memory to better compress a packet generated by a real-world content server?" In Sec. IV, it was shown theoretically that clustering is necessary to effectively utilize the memory in the proposed memory-assisted compression. Within this framework, we identify two interrelated problems: 1) How do we perform clustering of memorized packets to improve the memory-assisted compression in real-world traffic? 2) Given a set of clustered packets, how do we classify an incoming new packet (to be encoded) into one of the clusters in the memory using which the performance of memory-assisted compression is optimized?

In the sequel, we describe a hierarchical clustering algorithm that proves to be useful for compression. The proposed hierarchy for the content-aware joint memorization and clustering for network packet compression is shown in Fig. 3. As shown, we first identify whether or not an incoming packet is compressible. If the packet is determined incompressible, it is neither compressed nor stored in the memory. On the other hand, if a packet is determined compressible, it is passed to the clustering unit which operates based on the Hellinger distance metric.

Compressibility Determination: The compressibility determination is performed based on the empirical entropy of the data packet. The packets in memory may be divided into two categories: one category contains packets with very high entropy rate (close to 8 bits per byte) and hence these packets are incompressible. The other category contains packets whose empirical entropy rate is estimated to be much less than 8, and hence, these packets are compressible. Therefore, as the first step, the packets are partitioned into compressible and incompressible. After the partitioning step, the packets in the resulting memory are all compressible. Then, we will perform a clustering of the resulting memory based on the Hellinger distance metric between the packets.

A. Content-Aware Clustering Using Hellinger Distance Metric

The Hellinger distance is a metric to quantify the similarity between two probability distributions (cf. [13]). For two probability distributions $p(x)$ and $q(x)$, the Hellinger distance is defined as

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

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

1) Clustering: A good clustering is such that the packets clustered together should share similar statistical properties. Thus, they compress well together. Suppose the total number of clusters is given by $\mathcal{K}$, and each packet in the memory needs to be assigned to one of the clusters. We use the binary indicator c_t^j to denote the cluster assignment for the t-th packet $y^n(t)$. The indicator $c_t^j = 1$ if $y^n(t)$ is assigned to cluster $j \in [\mathcal{K}]$, otherwise $c_t^j = 0$. Then, the objective function for clustering is given by

$$J = \sum_{t=1}^{T}\sum_{j=1}^{\mathcal{K}} c_t^j d_H(q_t, u_j), \quad (2)$$

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

The problem setup suggests that the K-means clustering algorithm [14] is very suitable for our purpose. It is an iterative algorithm which consists of two steps for successive optimization of c_t^j (and hence u_j). Given cluster center u_j, the optimal c_t^j can be easily determined by assigning the packet $y^n(t)$ to the closest cluster with minimum Hellinger distance $d_H(q_t, u_j)$. Then, we fix c_t^j and update u_j.

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

$$c = \underset{1 \le j \le \mathcal{K}}{\operatorname{argmin}}\, d_H(q, u_j). \quad (3)$$

VI. Simulation Results

We demonstrate the effectiveness of the content-aware joint memorization and clustering proposed in this work through computer simulations. We apply our algorithm to real-world packets captured from CNN and Apple websites. To capture the packets, we have used *wget* and *wireshark* [15] open-source packet analyser together and stored the IP packets. We captured 8100 data packets, all with the size of 1434 bytes from each website for our experiment. One hundred packets are randomly selected from each website as test packets to measure the

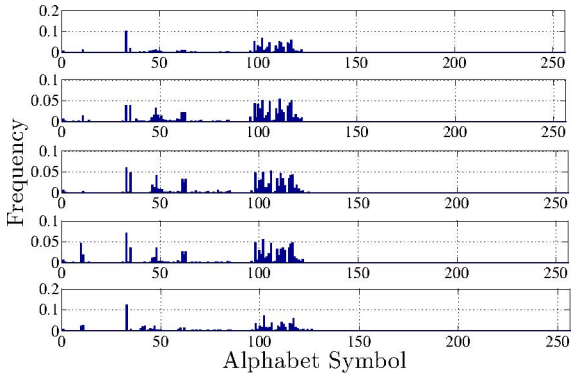

Fig. 4. Empirical symbol distribution of cluster centers.

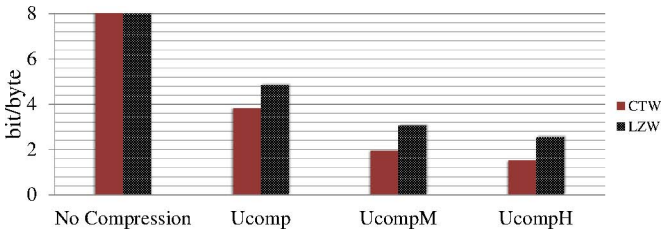

Fig. 5. Results of various memorization and clustering schemes on compressible packets after entropy classification.

compression performance. The remaining 16000 packets are then used to construct memory and the clustering algorithm is performed on these packets.

Fig. 4 demonstrates the probability distribution associated with each of the obtained clusters after the convergence of the clustering algorithm. As can be seen, clusters have significant amount of English alphabet as well as special characters that are attributed to either text or scripts. To illustrate the importance of clustering, we have evaluated the compression rate for the following cases after the entropy classification is performed. 1) Ucomp (Universal compression), 2) UcompM (Universal compression with naive memorization), and 3) UcompH (Universal compression with memorization and K-means clustering using Hellinger distance).

The simulation results are shown in Fig. 5. The compression rate on the compressible packets is plotted for the different packet coding strategies. As can be seen, our proposed clustering scheme based on the Hellinger distance metric achieves superior performance over the traditional universal compression. Note that these results do not demonstrate the impact of the incompressible packets. This is due to the fact that the number of incompressible packets may vary in different websites as some traffic traces contain more images and some contain more text/scripts. However, the trend shown here for the compressible sequences would remain intact for different traces. Furthermore, if we also consider the incompressible part of the data, the resulting compression rate would be slightly worse, but still considerably better than Ucomp and UcompM. This is indeed demonstrated in Table. I.

VII. Concluding Remarks

This work was motivated by our recent result that memorization can help to improve the compression performance of universal coding techniques for stationary sources. We extended the memory-assisted compression framework to accommodate the compound information sources, i.e., a mixture of stationary information sources behaving collectively as a content generator in the network. Our results indicate that by clustering the memorized packets from a compound source considerable performance improvement is achievable. We also presented a fast clustering algorithm tailored for the compression problem at hand. The memory-assisted compression has a complexity linear in the packet size, and hence, can be implemented real-time. The compressibility determination based on entropy estimation is also a linear operation in size of packets as it only entails the computation of empirical entropy of each packet.

The clustering algorithm applies the well-known K-means algorithm using the Hellinger distance metric. The K-means clustering is a more complex operation that needs to be performed offline. After the cluster centers are determined, which can be done every once in a while, the classification can also be done efficiently in linear complexity with the packet size.

Finally, we verified our theoretical results as well as the clustering algorithm via simulation of real Internet traffic from a mixture of CNN and Apple data servers. We observed that a factor of 2 improvement in compression is achieved over traditional compression using joint memorization and clustering.

References

[1] Z. Zhuang, C.-L. Tsao, and R. Sivakumar, "Curing the amnesia: Network memory for the internet, Tech Report," 2009. [Online]. Available: http://www.ece.gatech.edu/research/GNAN/archive/tr-nm.pdf

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

[3] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Info. Theory*, vol. 23, no. 3, pp. 337–343, 1977.

[4] F. Willems, Y. Shtarkov, and T. Tjalkens, "The context-tree weighting method: basic properties," *IEEE Trans. Info. Theory*, vol. 41, no. 3, pp. 653–664, May 1995.

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

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

[7] ——, "Memory-assisted universal compression of network flows," in *IEEE INFOCOM*, Orlando, FL, March 2012, pp. 91–99.

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

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

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

[11] E. Halepovic, C. Williamson, and M. Ghaderi, "Enhancing redundant network traffic elimination," *Computer Networks*, vol. 56, pp. 795–809, 2012.

[12] E. Zohar, I. Cidon, and O. O. Mokryn, "The power of prediction: cloud bandwidth and cost reduction," in *Proceedings of the ACM SIGCOMM 2011 conference*, ser. SIGCOMM '11. New York, NY, USA: ACM, 2011, pp. 86–97.

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

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

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