Improved Compression-Latency Trade-Off via Delayed-Dictionary Compression

Yossi Matias
School of Computer Science
Tel-Aviv University, Israel
matias@post.tau.ac.il

Raanan Refua
School of Computer Science
Tel-Aviv University, Israel
raananr@post.tau.ac.il

***Abstract*— We have recently introduced a novel compression algorithm for packet networks: *delayed-dictionary compression*, which enables an improved compression-latency trade-off. By allowing delay in the dictionary construction, the algorithm handles effectively the problems of packet drops and packet reordering: Its compression quality is close to that of streaming compression (and is substantially better than that of standard stateless packet compression) while its decoding latency is close to that of stateless compression (and is substantially smaller than that of streaming compression). In this Demo, we demonstrate the key ingredients of the new compression technique and show the effect of the dictionary delay on the effective bandwidth and on the decoding latency. We also demonstrate an effective file-transfer with a UDP carrier over the internet, using the Planet-Lab platform.**

I. Introduction

Consider dictionary data compression in packet networks, in which data is transmitted by partitioning it into packets, and dictionary-based compression is utilized. The goal in packet compression is to allow better bandwidth utilization of a communication line resulting in smaller amounts of packet drops, more simultaneous sessions, and a smooth and fast behavior of applications (see, e.g., [1]).

In dictionary compression, an input sequence is encoded based on a dictionary that is constructed dynamically according to the given text. The compression is done in a streaming fashion, enabling to leverage on redundancy in the input sequence.

In packet networks, packets may arrive reordered, due to different network characteristics, or due to retransmissions in case of dropped packets. Since streaming dictionary-compression assumes that the compressed sequence arrives at the decoder in the order in which it was sent by the encoder, the decoder must hold packets undecoded in a buffer until all preceding packets arrive. This causes *decoding latency*, which may be unacceptable in some applications.

To alleviate decoding latency, standard stateless packet compression algorithms are based on a packet-by-packet compression. For each packet, its payload is compressed, independently to other packets. While the decoding latency is addressed properly, this may often result in poor compression quality, since the inherent redundancy within a packet is significantly smaller than in the entire stream.

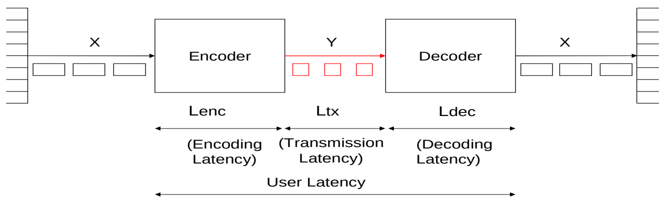

Fig. 1. *The end-to-end framework.* Two compression enabled network processors are used, one on each side of the communication link. The total user latency is the total of the encoding latency, transmission latency, and decoding latency. We focus on the decoding latency L_{dec} and the traffic compression ratio $r = |X|/|Y|$, where X is the original uncompressed traffic and Y is the compressed traffic.

A. Framework

The framework we consider is that of *end-to-end* compression over a communication link; see Fig. 1. We assume the existence of network processors, one at each side of the communication link. The original traffic that is to be transmitted through the communication link is now transmitted through the network processor and from there to the physical link. The network processors are *compression enabled*. By compressing the traffic a lower utilization of the communication line is obtained; equivalently, this results with effectively larger line bandwidth. This also results with a smaller number of packet drops, implying a smaller number of packet retransmissions. At this level we have packets carrying pieces of the original data which may be a stream of data, a file, or any other application information.

B. Potential Improvement in Packet Compression

As noted above, a packet-by-packet compression obtains improved latency, while compromising on the compression ratio. The objective is to maintain a similar latency, while getting the compression ratio as close as possible to that of streaming compression. The potential for improvement in compression is characterized as the ratio between the traffic compression ratio of stateless compression and the traffic compression ratio of streaming compression, denoted by φ. This ratio has been tested for various data files, for 125-byte packets; the results are shown in Fig. 2. If $\varphi > 1$ then we have room for improvement. For instance, for the file `rfc_index.txt` $\varphi = \frac{0.98}{0.44} = 2.23$, implying a potential improvement of over twice in the compression ratio. For

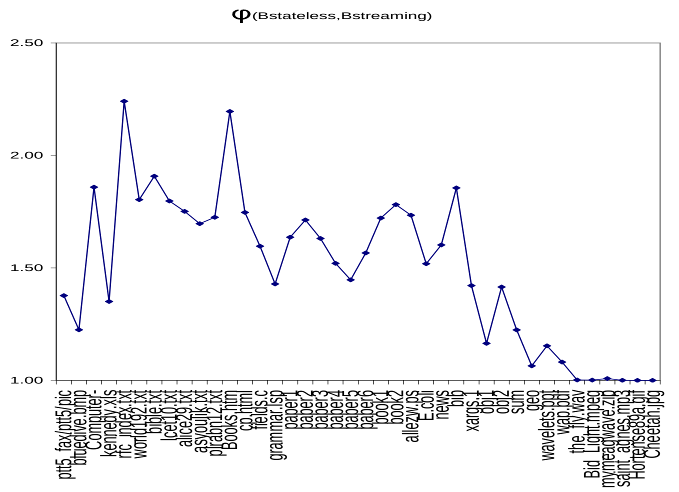

Fig. 2. The potential improvement φ for Calgary corpus files, Canterbury corpus files, and some files of our own. The payload size is 125 bytes.

already compressed files such as Cheetah.jpg, where $\varphi = 1$, there is no room for improvement.

II. Delayed-Dictionary Compression

We introduce in [2] a novel compression technique suitable for packet networks: the *delayed-dictionary compression* (DDC). The DDC is a general framework that applies to any dictionary algorithm. It considers the dictionary construction and the dictionary-based parsing of the input text as separate processes, and it imposes a delay Δ in the dictionary construction. As a result, when decoding a packet, the decoder does not depend on any of the Δ preceding packets, eliminating or diminishing the problems of out-of-order packets and packet drops compared to streaming compression, still with a good traffic compression ratio.

A full trade-off between compression ratio and decoding latency can be obtained, bridging between the extreme alternative of streaming compression (best compression ratio and worst decoding latency) and that of confirmed dictionary compression (same decoding latency as of stateless compression, yet better compression ratio). This trade-off is depicted in Fig. 3. Thus, the DDC has the benefits of both stateless compression and streaming compression. With the right choices of the dictionary delay parameter, it can have a decoding latency which is close to that of stateless compression, and with a compression ratio which is close to that of streaming compression.

The DDC method has an inherent conflict between the average decoding latency, $\overline{L_{dec}}$, and the traffic compression ratio: increasing the dictionary delay causes a decrease in the $\overline{L_{dec}}$, but increases r. This conflict is depicted in Fig. 4.

III. Experiments

We have tested the actual decoding latency of streaming compression by using transmissions of packets between pairs of Planet-Lab nodes, see Fig. 5. We have tested the traffic compression ratio of DDC compared to stateless compression

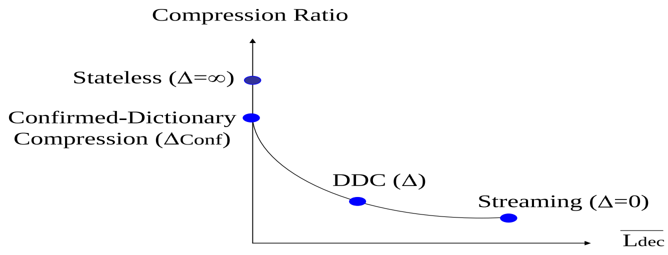

Fig. 3. The trade-off between the compression ratio and the decoding latency. Streaming has the best compression ratio and the worst decoding latency. DDC has compression ratio close to that of streaming compression, and also a decoding latency which is close to that of stateless compression. The *confirmed-dictionary compression* algorithm ensures a zero decoding latency, as in stateless compression, yet improved compression ratio.

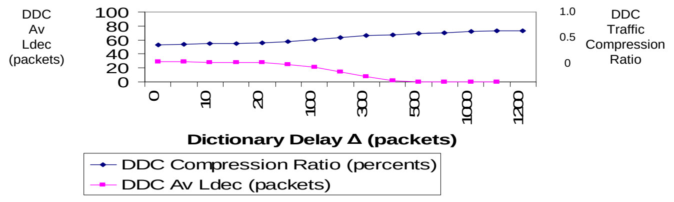

Fig. 4. DDC Conflict: The upper part of the figure is a plot of the traffic compression ratio in the DDC method as a function of the dictionary delay, e.g., for a dictionary delay of 1000 packets the traffic compression ratio is $r = 0.72$. The lower part of the figure is a plot of the average decoding latency of DDC, $\overline{L_{dec}}$, in terms of packets; e.g. for a dictionary delay of 300 packets, the average decoding latency is 7.8 packets.

and streaming compression. The traffic compression ratio for each delay value for a version of DDC called DDC-min is depicted in Fig. 6. The payload size is 1500 bytes. The traffic compression ratio for stateless compression and streaming compression also appears in this figure.

We have tested the DDC method using actual transmissions over the internet, using the Planet-Lab testbed [3]. The distribution of decoding latencies of 6,997,228 packets sent over a transmission line with round-trip time (RRT) of 5000msec, is depicted in Fig. 7. We show the distribution for both streaming compression as well as for DDC with a delay of $\Delta = 300$. In streaming (lower plot), the percentage of pending packets, i.e, packets with $L_{dec} \neq 0$, is 18.8% with maximal L_{dec} value of

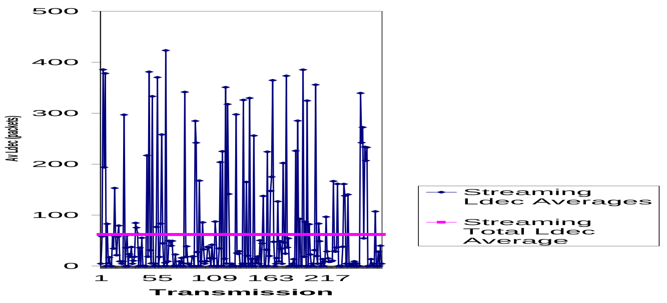

Fig. 5. The average decoding latency of streaming for every transmission out of the 269 transmissions over Planet-Lab. The variance in terms of packets is very high. The total average is 62 packets, represented by the horizontal line.

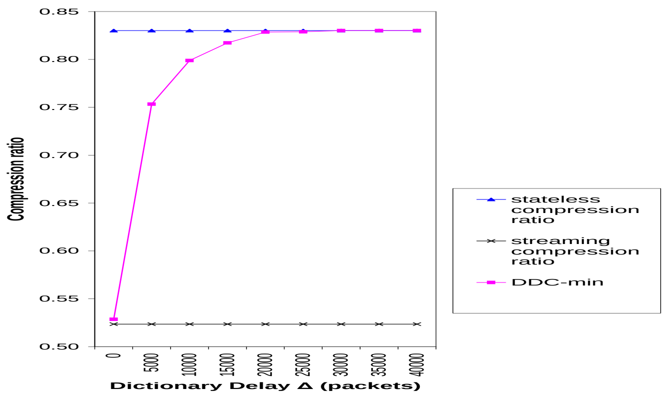

Fig. 6. Compression ratio of DDC-min as a function of the dictionary delay in packets, compared to stateless compression ratio and to streaming compression ratio. The data file in use is the concatenation of 18 Calgary corpus files, $|Header = 20|, |Payload| = 1500$. DDC-min is useful for Δ of up to 1500 packets.

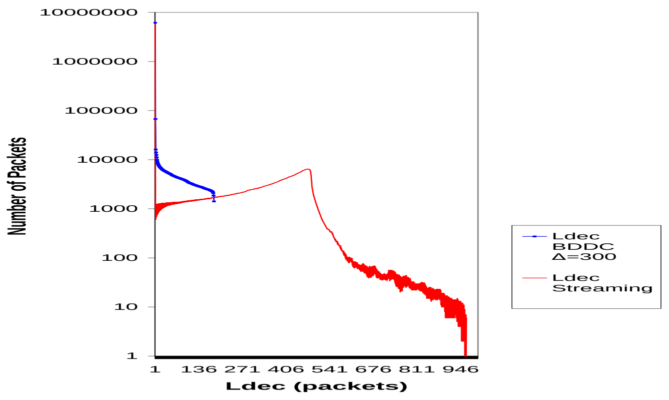

Fig. 7. The Decoding Latency Distribution of Streaming compression (lower plot) and DDC with $\Delta = 300$ (upper plot), for transmission over a line with $RTT = 5000$msec. The maximal L_{dec} value for streaming is $4RTT$ which is 963 in terms of packets. The maximal L_{dec} value for DDC is $2RTT - \Delta$ which is 383 packets. DDC achieves substantially better L_{dec} distribution compared to streaming.

963 which represents $4RTT$. [1]

In the case of DDC, the percentage of pending packets is reduced to 17.1%. The maximal L_{dec} for DDC is 383 packets which represents $2RTT - \Delta \geq 0$. Overall, the decoding latency distribution of the DCC is substantially better than that of streaming compression.

IV. An Example Application

The DDC is mostly applicable to applications where both bandwidth and latency are of importance, and where the order between decoded packets is not essential.

An example application is massive utilization of instant messages. Another application may be the monitoring-related messages for network management applications. In both cases latency is important, where the order between decoded packets may not be significant.

[1]In case of a massive packet drop that comes in a large burst which is equal to $2RTT$ in terms of packets, the first non-dropped packet after the burst may wait up to $4RTT$ since the retransmission timer is set to $2RTT$.

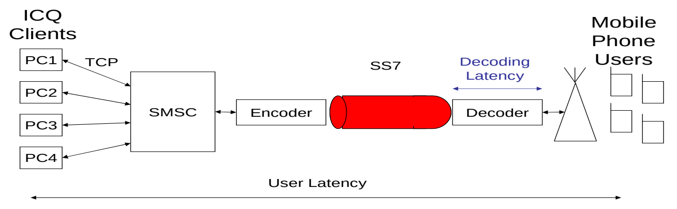

Fig. 8. SMS Chat from PC to Mobile: SMS messages are sent from Instant Messenger PC clients to mobile phones by using TCP at first, and SS7 links afterwards. The bottleneck is the low speed SS7 links.

We describe in more detail an application of PC-to-mobile chat over SMS (Short Message Service) messages. Such application exists in PC based instant messenger clients (e.g., ICQ), which enables sending SMS messages to mobile phones. The SMS messages are transferred to the SMSC (Short Message Service Center) by using special SMS protocols. Then the SMS messages are transferred to the mobile phones through dedicated low speed SS7 (Signaling System 7) control links. The speed range of the SS7 control links is 56kbps to 64kbps, therefore these links are considered a bottleneck. The system overview of this application appears in Fig. 8.

The DDC method enables improved traffic compression over the SS7 links. The compression enables more SMS messages to be sent along a channel with a given bandwidth, with good decoding latency.

V. Demo Scenario

We demonstrate the key ingredients discussed in the companion paper [2]:

• The effect of the dictionary delay on the compression ratio. We demonstrate the traffic compression ratio of stateless compression and streaming compression simultaneously to the one of DDC.

• The effect of the dictionary delay on the decoding latency. We show that larger dictionary delay improves the decoding latency, and demonstrate the decoding latency of stateless compression and streaming compression simultaneously to the one of DDC.

• DDC-enabled communication over the internet, using Planet-Lab [3] nodes. We demonstrate a file-transfer using DDC with UDP as carrier over the internet, as in TFTP [4]. TFTP is widely used in networking applications, especially where user authentication and directory visibility are not required. In this part of the demo we will see the effect of the dictionary delay on the traffic compression ratio, on the dictionary delay, and on the total transfer time which is better than the one of streaming compression.

References

[1] C. Westphal, "A user-based frequency-dependent IP header compression architecture," *IEEE, Globecom*, 2002.
[2] Y. Matias and R. Refua, "Delayed-dictionary compression for packet networks," *IEEE, Infocom*, 2005.
[3] "Planet-lab," http://www.planet-lab.org.
[4] K. Sollins, "The TFTP protocol," *IETF, RFC 1350*, 1992, http://www.ietf.org/rfc/rfc1350.txt.