http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 12 (24): 8109-8115, 2013
ISSN 1812-5638 / DOI: 10.3923/itj.2013.8109.8115

A Real-time Data Compression for Massive Data Transmission Over High Delay-bandwidth Network

Zheng Chen, Chong Peng and Long Qin
School of Computer Science and Engineering,
University of Electronic Science and Technology of China, 610054, Chengdu, China

Abstract: Packet dropout and retransmission have become a more and more serious problem in high delay-bandwidth network, it is difficult for traditional streaming compression to deal with reorder packets, people have to use stateless compression, but it can only achieve very low compression ratio. In this study, we introduce a novel Orderless Tolerance Compression Algorithm (OTCA). The algorithm works effectively with orderless packets which is caused by the packet retransmission, through allowing variable delay in the dictionary construction. OTCA performs better compression ratio than stateless compression and low decoding latency than that of streaming compression and excels delay-dictionary compression in both compression ratio and decoding latency especially in high bandwidth network. We conduct extensive experiments to establish the potential improvement for packet compression techniques, using many data files including the Calgary corpus and the Canterbury corpus. Experimental results of the OTCA show that it is a good compromise proposal for transfer massive data over high delay-bandwidth networks.

Key words: Orderless tolerance compression algorithm, OTCA, high delay-bandwidth network, data compression

INTRODUCTION

Information becomes extremely important to their company, in order to protect them from being destroyed by sudden disasters such as flood, earthquake, stolen, most companies backup their information remotely from time to time, therefore, a large volume of data need be transferred over the Wide Area Network (WAN). However, WAN is characterized with high bandwidth, high delay and unstable at present days, resulting in high packet loss rate and data retransmission.

A lot of approaches have been proposed to speed up data transmission over the packet network, on one hand, people improve network bandwidth with new data transmission protocols, such as UDT, SABUL and XCP. On the other hand, application reduces size of data through de-duplication, binary caching and compression. De-duplication detects and eliminates the duplicate copies between data server and its backup, binary caching reduce the repeating data over network, however, data compression deal with file or streaming data in a smaller granularity, it works in the bit level, encodes information using exploit statistical redundancy to represent data more concisely without losing information. With the benefits of these three technologies, a remote backup application can highly reduce the packet over network.

Since, the character of WAN, packets arrive out of order at the backup-end, but a packet could not be decompressed if its prior packets are still on the way, thus lead to a decompression delay. It becomes an open problem to solve the reorder packets in the network compression (Jeannot and Knutsson, 2002). Stateless compression compresses each packet separately (Friend and Monsour, 1998; Pereira, 1998; Rand, 1996; Tan *et al*., 2010), so it can ignore the disorder of packets, but results a fatal weakness of low ratio (Borella, 2000). Streaming compression cannot decompress any reordered packet, for each packet has to decompress by the phrase of its preceding packets, it has the best ratio but do not suitable for real time application (Ziv and Lempel, 1977; Matias *et al*., 2001; Dorward and Quinlan, 2000). Delay Dictionary Compression (DDC) (Matias and Refua, 2005) can decompress reordered packets by adding a Δ delay in the dictionary construction, benefitting for low decompression latency, it works inefficient when the delay becomes serious in the network. Other methods such as IP header compression and bulk compression (Huffman, 1952) which aggregate the same resource and destination packets then replace the redundant portions with shorter codes at gateway do not take reorder into consideration, so they can't be deployed in the real-time application. In order to deal with the high delay network,

Corresponding Author: Zheng Chen, School of Computer Science and Engineering,
University of Electronic Science and Technology of China, 610054, Chengdu, China

we proposed a new algorithm called Orderless-Tolerance Compression Algorithm (OTCA). OTCA adds a variable rather than constant Δ delay in the dictionary construction Therefore, the decoder side can decompress almost all the packets which arrive reordered. With this method, OTCA provides closed compression ratio to streaming compression and better decoding latency comparing to DDC.

The rest of the study is organized as follows: Section 2 discuses the detail of state of art. Section 3 illustrates OTCA in detail, including the algorithm principal and implementation. Experiments and results are shown in Section 4, represents the benefits with our method comparing to other algorithms.

RELATED WORK

Many protocol-independent lossless data compression algorithms have been used in packet network, those can be divided into two types, statistics-based algorithm (Huffman, 1952) and dictionary-based algorithm (Lilley *et al*., 2000; Welch, 1984; Pereira, 1998). Statistics-based compression algorithm compresses data according to the frequency of each letter, encoding high frequency data into fewer bits to reduce the volume of data. Statistics-based compression has higher compression ratio, but need read data twice which resulted in higher complexity and delay. Dictionary-based compression algorithm uses previous data as dictionary, substitutes a reference to the same phrase in the dictionary to replace the original data. New phrases that not exist in the dictionary will be added into the dictionary and compression work continues until the whole data has been processed. Dictionary-based compression algorithms performs faster compression speed than statistics-based dictionary, also need more resources in some application scenarios.

By how to use the above algorithm, the application of compression can be divided into stateless compression, streaming compressions, off-line compression and delay dictionary compression. They can use either of these above compress algorithm, but almost all the practical applications are based on dictionary compression algorithm, because of its higher performance, so we'll only discuss dictionary compression algorithm the rest of this study.

Packet-based stateless compression is the most simple and straightforward one. In packet-based compression, each packet is compressed independently, the history is initialized when each packet has been compressed and decompressed. At the receiver, each packet can be decompressed, regardless of the packet drops and order arrival. Packet-based compression takes relatively considerable resources, but gives poor compression ratio.

In streaming compression the dictionary is initialized before each packet is encoded (same at the receiver). The dictionary is constructed by the preceding packets and the current packet itself. More and more phrases are added into the dictionary. In this method each packet is encoded in order. At the receiver, if one's logic prior packet is missing, the application should store the current packet until all the prior packets received through retransmission, resulting in large values of decompression latency and memory resources.

Off-line compression is a solution with less resources, encoder breaks compressed data into packets and sends one by one when the compression works finished. Decoder decompresses packets in order after every one has arrived. This approach works at the application not at the network processor inside, so the network processor need a bit of memory resource. Since, the network processor treats a packet as standard packet, it just breaks arrival data into packets and sends them or receives each packet and deliveries them in order to application layer. Decoder decompresses them without any latency compared previous depicted compression algorithms. Off-line compression algorithm provides good compression ratio even better compared to streaming compression algorithm because it can unions several different compression algorithms to deal with data.

Delay dictionary compression is the only solution that considered the unstable of network. In order to make a compromise on low compression ratio in packet-base compression and high latency in streaming compression, DDC imposes a delay Δ in the dictionary construction, as Fig. 1 shows, each packet is compressed by using the front of its Δ delay packets. As a result, DDC eliminates or diminishes the reorder packets and packet drops. DDC has provides four encoding algorithms, Adaptive Delay Algorithm (ADA), Confirmed-Dictionary Compression Algorithm (CDCA) and two alternative algorithms for DDC. ADA detects the delay at decoder side every small period of time, increases (resp. decreases) Δ when decoding delay becomes bigger (resp. smaller), the compression ratio changes responsibly, small Δ result in good compression ratio. CDCA uses packets that have already arrived at decoder as compression dictionary, so the decoder can decompresses each packet immediately, The decoder does not need store any out-of-order packets or packet drops, resulting resources saving. One alternative algorithm for DDC compresses packet twice-by the BDDC (basic DDC) dictionary and packet dictionary, the result will be the minimum between the uncompressed

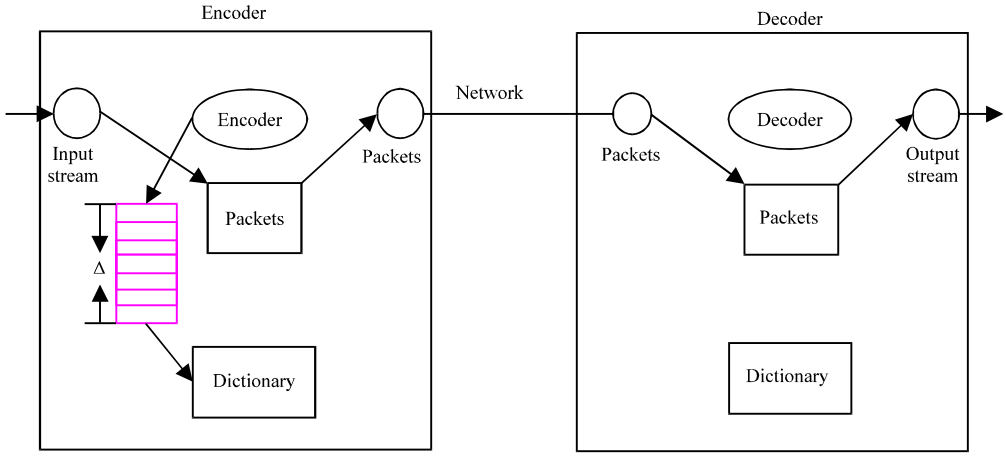

Fig. 1: DDC basic ideas: There is a queue that holds Δ packets, once a packet is compressed, the encoder gets the oldest packet from the queue and reconstruction the dictionary (resp decoder)

lengths and the packet compression length and the BDDC compressed length. DDC solves the problem of reordered packets that streaming compression can't process and performs good compression.

ORDERLESS TOLERANCY COMPRESSION ALGORITHM

Compression ratio and decoding latency: Let X be the size of uncompressed data, Y be the size of compressed data and r = Y/X will be the compression ratio. Generally, the compression ratio is between 0.0 and 1.0, the smaller r, the better.

Let r_A be the compression ratio of A algorithm, $\phi = r_A/r_B$, ϕ is the ratio of compression ratio of A and B, if ϕ is bigger than 1, the B algorithm performs better than A algorithm, otherwise, A algorithm is worse than B.

Let i be the serial number of packet, Δ_i be prior packets of i that wait to be decompressed at the decoder, Ω be the total number that decoder send. $\delta_{delay} = (\Sigma\Delta_i)/\Omega$ is the average delay at the decoder.

Orderless tolerancy compression algorithm: In dictionary compression, a specific dictionary of a compressing phrase is also called a look-ahead window. The dictionary needs to be updated with new phrases joining. If the current phrase that compression program just finishes reading is inserted to the dictionary, the dictionary construction is called zero delay updating; otherwise, it is called delay dictionary. Let non-negative Δ be the delay in the dictionary construction, n be the serial number of current phrase. Therefore, the look ahead window is made up by the front of n-Δ-1 phrases. Generally, Δ is 0 in traditional dictionary compression. If the compressed data have to transferred over unstable network, set the delay in the look-ahead window can lower the decoding latency caused by retransmission and packet drops and how to get a properΔis the key of a network compression.

If we consider it more specifically in a network compression, the Δ will be the delay of packets, we have to use several bytes (in this case, we use 4 bytes) of each the header of packet to record the Δ. And because network compression is always Protocol Independent, if the data is multimedia data, like Mp3 or Jpeg, it may never be compressed by dictionary at all, so, if the uncompressed data is less than the sum of compressed data and the extra length (at least 4 bytes), the encoder transmits the original packet without compression:

- **Encoder:** We use a FIFO queue to store the information of encoded packets which is empty before data compressing, each element in queue includes the serial number, beginning and ending position in the original data. According to the OTCA dictionary updating mechanism, there are two stages in the dictionary construction: at the first stage, OTCA uses stateless compression to encode each packet like DDC, the dictionary is useless for the next packet and therefore, each packet constructs its own dictionary. There is no delay in the dictionary. The second stage is completely different from the first one. In the second stage, the dictionary is still useful for the next packet, but the phrase in current packet do not be inserted to the dictionary any more. As discussed before, Δ is variable. Suppose the upper

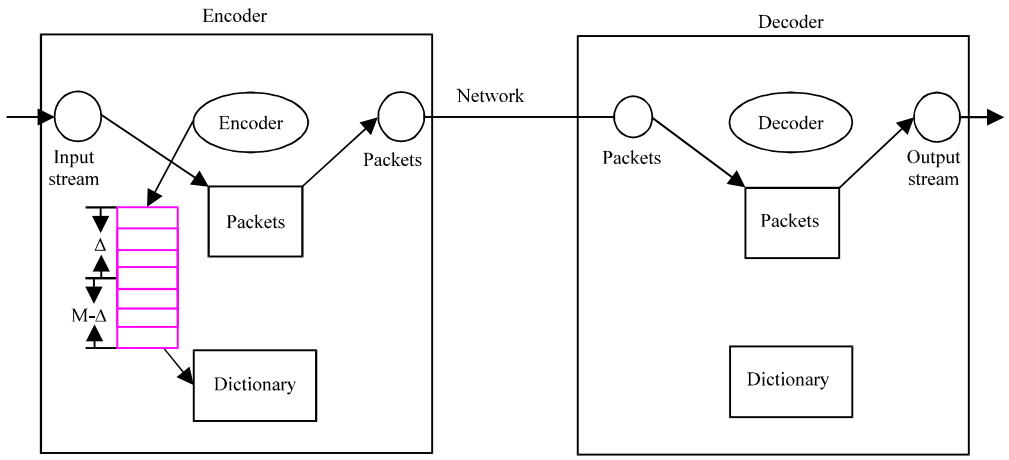

Fig. 2: We set additional M-Δ caching packets for data compression. if the link status is good, the dictionary doesn't need update as soon as one packet has been processed and send, making full use of the network rather than compression

limit is M, OTCA acquires the front M-Δ packets, Fig. 2, to update dictionary when Δ has reached its upper limit

We use network speed measurement tool to obtain the link status between encoder and decoder. If the network is busy and packet drops is serious, OTCA keeps Δ stable, thus, one packet is inserted to the queue, OTCA extracts a packet from the queue and insert the phrase into dictionary. When the communication link between encoder and decoder is fine, OTCA updates dictionary until the queue is full. In DDC, when one packet has been inserted to the queue, the dictionary updates once. It does not consider the link status of the network, resulting in high decoding latency in decoder. Although adaptive delay algorithm in DDC let decoder report the decoding latency to encoder and the encoder changes the Δ dynamically according to the decoding result, however, the result is delaying, it cannot reflect the real situation of the network

- **Decoder:** Each packet may arrive at decoder through network out of order, the decoder check its serial number. If the serial number minuses serial number of the last packet in the dictionary equaling to the Δ delay that contained in the packet header, the decoder inserts the received packet in the ready queue, otherwise, inserts it in the waiting queue. The decoded packet will be inserted to a special queue (called decoded queue) like the encoder for constructing dictionary

To avoid premature dictionary update, we build a function that examines whether all the packets which depend on this dictionary are decoded. If dictionary need update, OTCA acquires the front packets for the decoded queue, inserts all the phrases to the dictionary.

Pseudo code of OTCA: As we discussed above, each packet is compressed with a variableΔdelay. The dictionary will refresh when the serial number of current packet has be out of bounds. To make better understand of OTCA, the pseudo code has been describes as following:

Encoder Variables and fuctions

ps: *the pointer of current streaming data*
streaming_end: *the end address of a streaming*
getTrait(ps): *get the trait of 4 sequential bytes which start at ps.*
DIC: *compression dictionary*
getRepeatedBytes(): *get the address of repeat bytes in DIC*
encode(ps, len): *compress the len bytes start at ps into a packet*
delta: *the interval between current packet and the last packet of the DIC.*
needCompressNewPacket(): *check whether compress anew packet.*
sendCurrentPacket(): send well compressed packet.
increaseNumByOne(): increases the serial number of the next packet by one
updateDIC(lastPacketOfDIC, Δ): update the dictionary with next Δ packets from the last packet of dictionary

Pseudo code of OTCA encoder

```
while ps<streaming_end
 if needCompressNewPacket()
 sendCurrentPacket();
 if Δ< delta<M //do not need update DIC
 increaseNumByOne();
 continne;
 else
 updateDIC(lastPacketOfDIC, Δ);
 end if
 if getTrait(ps) in DIC
 getRepeatedBytes();
 encode(ps, len);
 end if
```

Decoder variables and fuctions

receivePacket(): receive packet from network
hasPacket(): check whether there are packets
getTopPacket(): get the top packet in the queue
getSerialNum(packet): get the serial number of a packet
decode(packet): decompress a compressed packet
cnt: the counter of how many packets have been decoded
insertIntoQueue(): insert a packet into the waiting for decompressing queue

Pseudo code of OTCA decoder

```
receivePacket();
while hasPackets()
packet = getToppacket();
delta = getSerialNum(packet)
if Δ<delta<M
decode(packet);
cnt++;
else
insertIntoQueue(packet);
end if
if cnt>M-Δ
updateDIC(lastPacketOfDIC, Δ);
cnt=0;
else
continue;
end if
```

EXPERIMENTS

The above algorithm is been implemented in C^{++}, based on Windows 7 OS. To test its practical properties, we use Netem network simulator running on Linux to simulate a high delay-bandwidth but unstable network. We set the network to 100 Mbps, 250 m sec delay and 1% packet drop rate, test the compression ratio and average uncompressing latency. We implemented DDC_min too which also set a delay in dictionary and compare the performance of these two algorithms.

Compression ratio: In this test, we focus on correlation between delay and compression ratio. Streaming compression has the best compression ratio without any delay in dictionary, so we roughly think that the delay in the dictionary has effects on compression ratio. Figure 3 shows the relation between delay and compression.

Basically, stateless compression and streaming compression show the high and low bound of compression ratio. Both OTCA and DDC's compression ratio are better than streaming and less than stateless, they has very close performance in any condition.

Comparing to stateless compression, none of these two algorithms can achieve close results and we can do less work in data compression with the delay increasing, because the relative information are missing, so we must find a proper delay to obtain the relative low compression ratio.

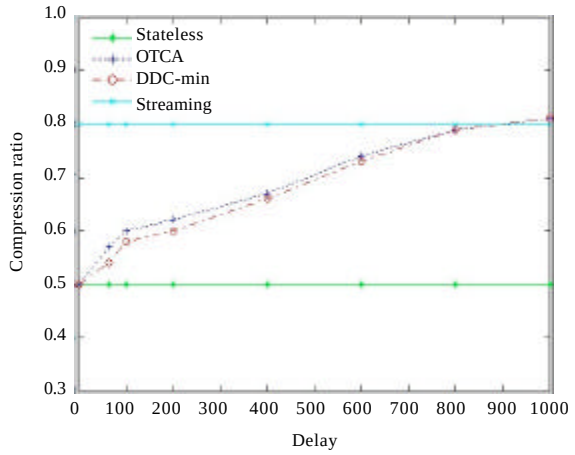

Fig. 3: Trade-off between delay in dictionary and compression ratio

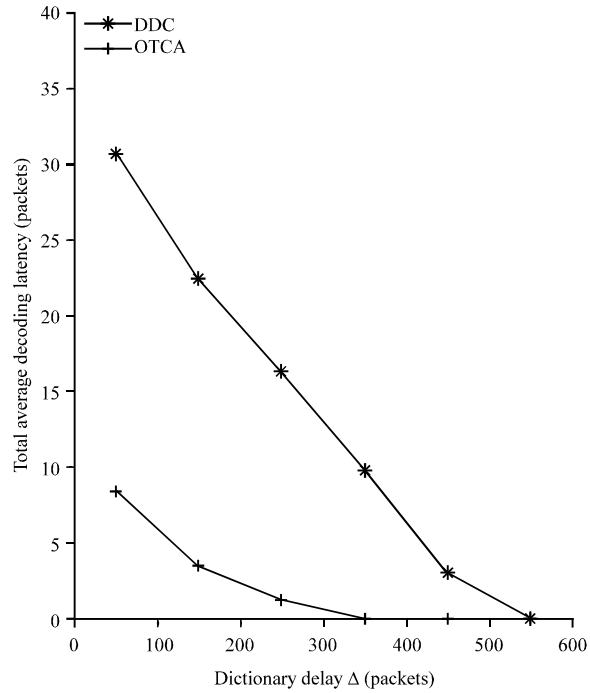

Fig. 4: Total average latency and dictionary delay Δ(packets): In this experiment, M-Δ is 200 packets. RTT = 5000 msec, OTCA is less half average decoding latency comparing to DDC

Average latency: In a reliable transfer protocol, each lost packet will be resent if the ACK can't be received. It costs two RTT to retransmit the lost packet, leading to latency in decoding packet if we use streaming compression or other similar compression algorithm. Figure 4 shows the relation between average latency and delay. We do not

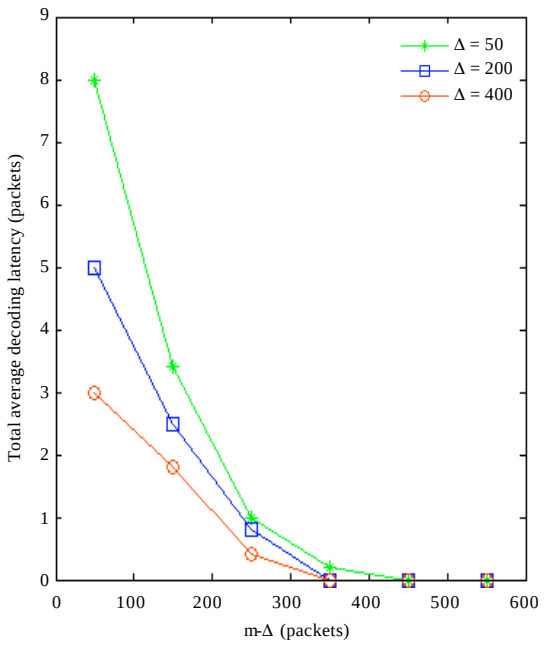

Fig. 5: OTCA with different M-Δ packets: the average decoding latency decreases quickly with the parameter increasing

show the stateless compression and streaming compression, because one will be total zero and the other will be far out of the Y-axis.

We use a hash container to store the packets that come from the network, making sure each packet can be decompressed when the decompression dictionary is constructed. In order to lower the average decoding latency, DDC provide two dictionaries, one for theΔdelay packets and the other for the just arriving ones. DDC looks more delicate, but our OTCA has performance much better than them.

Variable parameter: The OTCA parameter is very useful in lowering average decoding latency, as Fig. 5 depicts, when the m-Δis greater than 350 packets, the latency is close to 0. However, the OCTA parameter has effects on the compression ratio, just as Fig. 5 shows, the delay between current processing packet and the dictionary almost decides the compression ratio of DDC and OCTA. Actually, it works in every dictionary based compression algorithm.

With our method, the compression ratio of OCTA is very close to DDC, but the less average decoding latency we get. Since, the network has limited transmission speed, the delay in dictionary can be relative small. In our method, when the delay is 50 packets, we can get the best balance between compression ratio and decoding latency and the compression ratio is close to streaming compression.

CONCLUSION

In this study, we introduced how to transfer massive data over WAN, after have a glace of how to reduce data, we focused on the popular methods that have been used in network compression, including stateless compression, streaming compression and delay dictionary compression. Then we propose a new method compression algorithm OTCA to lower the average decoding latency through introducing a variable delay in dictionary.

Since, there are amount of data should be transmitted over the network every second, the network transmission optimization attracts many researchers. In the future, there are more dispersed data produced by personal user should be transmitted over the network with the demands of less transmitting time and network optimization for massive data can go more far away.

ACKNOWLEDGMENT

This research was supported by the National Natural Science Foundation of China (Grant No. 61003034) and the Huawei innovation funds.

REFERENCES

Borella, M.S., 2000. Measurement and interpretation of internet packet loss. J. Commun. Networks, 2: 93-102.

Dorward, C. and S. Quinlan, 2000. Robust data compression of network packets. Bell Labs, Lucent Technologies. http://www.cs.bell-labs.com/cm/cs/who/seanq/networkcomp.pdf

Friend, R. and R. Monsour, 1998. IP payload compression using LZS. RFC 2395, December 1998. http://www.ietf.org/rfc/rfc2395.txt

Huffman, D.A., 1952. A method for the construction of minimum-redundancy codes. Proc. Inst. Radio Eng., 40: 1098-1101.

Jeannot, E. and B. Knutsson, 2002. Adaptive online data compression. Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, July 23-26, 2002, Edinburgh, Scotland, UK., pp: 379-388.

Lilley, J., J. Yang, H. Balakrishnan and S. Seshan, 2000. A unified header compression framework for low-bandwidth links. Proceedings of the 6th Annual International Conference on Mobile Computing and Networking, August 6-11, 2000, Boston, MA., USA., pp: 131-142.

Matias, Y. and R. Refua, 2005. Delayed-dictionary compression for packet networks. Proceedings of the IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies, Volume 2, March 13-17, 2005, Miami, FL., USA., pp: 1443-1454.

Matias, Y., N. Rajpoot and C. Sahinalp, 2001. The effect of flexible parsing for dynamic dictionary-based data compression. J. Exp. Algorithmics, Vol. 6. 10.1145/945394.945404

Pereira, R., 1998. IP payload compression using DEFLATE. RFC 2394, December 1998. http://www.ietf.org/rfc/rfc2394.txt

Rand, D., 1996. PPP predictor compression protocol. RFC 1978, August 1996. http://www.ietf.org/rfc/rfc1978.txt

Tan, L.S., S.P. Lau and C.E. Tan, 2010. Enhanced compression scheme for high latency networks to improve quality of service of real-time applications. Proceedings of the 8th Asia-Pacific Symposium on Information and Telecommunication Technologies, June 15-18, 2010, Kuching, Malaysia, pp: 1-6.

Welch, T.A., 1984. A technique for high-performance data compression. Computer, 17: 8-19.

Ziv, J. and A. Lempel, 1977. A universal algorithm for sequential data compression. IEEE Trans. Inform. Theory, 23: 337-343.