SDP Compression Algorithm for WebRTC: ESDiPi

Adem Atalay*, Doğaç Başaran
Netaş Telecommunication A.Ş. Kurtköy-Pendik, Istanbul, Turkey.

* Corresponding author. Email: adema@netas.com.tr
Manuscript submitted October 17, 2015; accepted Feburary 10, 2016.
doi: 10.17706/ijcee.2016.8.1.77-83

Abstract: In Voice over IP or WebRTC technologies, the communicating parties use Session Description Protocol (SDP) for negotiation of the session and media capabilities to establish the connection. A typical SDP is a simple text data that has a length of approximately 3000 bytes however this size is not so suitable for some applications i.e. mobile applications. Standard compression algorithms such as G-Zip, 7-Zip, are usually applied to reduce the SDP sizes. Such approaches do not consider the specific structure of the data to be compressed hence better compression ratios aretheoretically possible. In this work, we propose a compression algorithm based on Lempel-Ziv (LZ77) lossless coding scheme combined with SDP packet structure. Experimental results reveal that our approach reaches much better compression ratios than some widely used compression algorithms.

Key words: WebRTC, session description, compression, Lempel-Ziv.

1. Introduction

Session description Protocol [1] (SDP) is a general purpose standard that is used in a wide range of network environments and applications such as multimedia applications and voice-over-IP calls. The main purpose of an SDP packet is to arrange a negotiation between communicating parties where the services to be supported (audio, video etc.) are determined. It is basically a simple text data that consists of the description of multimedia sessions such as transport addresses and other session metadata informationand media capabilities of communicating parties.

The session description data is transferred via several signaling mechanisms. Long-polling and HTTP Streaming [2] are two popular methods where Representational State Transfer (REST) architecture is used for signaling requests. In these methods, the HTTP Request body in the SDP packet is usually compressed with DEFLATE algorithm [3] which is very popular due to high compression rates. Such compression methods are necessary especially under circumstances where the channel has not enough bandwidth. There are also other SDP transfer methods where the SDP is not compressed such as SIP over WebSocket [4], [5].

In addition to those mechanisms, push notifications can also be used to transfer SDP packets in mobile communication systems. Other signaling schemes such as Long-polling and WebSocket have a high network overhead on the signaling server. A typical http request and its response have approximately 800 bytes overhead. In addition, WebSocket has 2 bytes in a message [6]. As a simple example, if 1,000 clients receive one message per second on a WebSocket mechanism, network throughput will be 16,000 bits per second. This throughput is reasonable for a standard network however its energy consumption on a mobile device is not efficient since the connection has to be alive as long as the WebSocket is open. This is achieved by ping- pong messaging even if no message received or sent over WebSocket. On the other hand, if 1,000 clients poll every second on a Long-polling connection, network throughput will be more than 800,000 bytes per second (more than 6,4 Mbps) that produces high network traffic.

The limit of a push notification message size changes according to the operating system. The maximum size allowed for a notification payload is 2 Kbytes and 4 Kbytes for iOS and Android respectively (Prior to iOS 8, the maximum payload size is 256 bytes) [7], [8]. Any notification that exceeds these limits is refused by the cloud system.The problem with transferring SDP packets usingPush Notification systems occurs due to these size limitations.A typical SDP packet length is approximately3Kbytes however under certain situations, an SDP packetmay reach up to 6Kbytes according to number of networkinterfaces such as Wi-Fi, GSM, VPN etc. Therefore injectingan SDP packet into a Push Notification message usually requires compression. It is possible to compress SDP packets in Push Notifications by applying DEFLATE algorithm as in HTTP Request body. However, higher compression ratios might be needed in certain situations. This is due to the fact that Push Notification payload contains not only SDPs but also user specific information. To be able to increase the space for user specific information, compression ratio must be increased as much as possible.

DEFLATE algorithm achieves high compression ratios using two legacy compression algorithms, Huffman coding [9], [10] and Lempel-Ziv (LZ77) [11] and it is widely used in text compression purposes. However, it is a general-purpose compression algorithm where the structure of the data is not taken into account. By designing a compression algorithm that is unique to a proper structure naturally achieves higher compression ratios than general-purpose methods. In this work, we propose such an algorithm where we apply Lempel-Ziv (LZ77) coding using the specific structure of the SDP packets.

The rest of the paper is arranged as follows: In Section 2, the DEFLATE algorithm, LZ77 algorithm and Huffman coding are described. Then, in Section 3, the proposed algorithm is introduced. The experimental results on different size SDP packets and comparison with DEFLATE algorithm are given in Section 4. Finally in Section 5 the conclusion and some future directions are given.

2. DEFLATE Compression Algorithm

Compression is defined as, the process of coding that will effectively reduce the total number of bits needed to represent certain information [12]. Both lossy and lossless compression algorithms exist in literature depending on the application. In our area of interest where the HTTP Requests/Responses or in general SDP packets are to be compressed, the compression have to be reversible i.e., lossless. A loss of information breaks the negotiation between communicating parties hence prevent a quality communication. One of the most popular lossless compression algorithms that is widely used in these situations is the DEFLATE algorithm. Beyond HTTP Requests, DEFLATE algorithm is also used in PNG images and file compressions like gzip [13], pkzip etc.

DEFLATE algorithm is based on the combination of two well-known lossless compression schemes, Huffman and Lempel-Ziv coding. Basically, the data is compressed with Lempel-Ziv coding and resulting pairs of data is further compressed using Huffman coding similar to JPEG compression algorithm in image coding [14]. Each data block begins with 3 header bits. The most significant bit represents whether the block is the final block or not. The subsequent two bits represent the type of the data block. There are three main types of blocks in the compressed data with DEFLATE algorithm;

1) Uncompressed data (00)
2) Compressed data with fixed Huffman codes (01)
3) Compressed data with dynamic Huffman codes (10)

A fixed (pre-agreed) Huffman tree is used in the first type whereas in the second type the Huffman table has to be supplied. Compressed block types might have arbitrary lengths on the other hand uncompressed data blocks havea size limitation of 65,535 bytes.

There is a basic difference in applying Lempel-Ziv and Huffman methods in the DEFLATE algorithm. Lempel-Ziv coding utilizes the whole data regardless of the packetsequence. On the other hand, only the data in the current block is used for Huffman coded blocks therefore theseblocks are independent from any other block. If the block type requires dynamic Huffman codes, the compressedblock consists of the Huffman table concatenated with the compressed data with Huffman codes. The precoded (before Huffman coding) data consists of literal bytes (charactersthat are not coded with LZ77) and *<length-distance>*pairs. Note that one code tree is used for literals andlengths and a separate code tree is used for distances. Thecode trees for each block appear in a compact form justbefore the compressed data for that block [3]. It is importantto mention that there is also a limitation of 32Kbytesas the maximum distance to reference position in LZ77coded pairs.

3. SDP Compression Algorithm: ESDiPi

Data compression provides a way to transmit or store same amount of data with fewer bits. Meaningful text data are the most compressible data in computer science because of the redundancy in the data. Redundancy in a text data can be expressed as entropy of characters or substring repetitions. Codes for representing some data is determined according to these redundancies. In general purpose compression algorithms the codes and the corresponding data must be given to decoder to know the meanings of the codes. This leads that some data must be left as uncoded or generation of code and data table is necessary. Therefore, discarding this uncoded data or coding tables make sense for special cases such as session description protocols. For this kind of cases instead of transferring uncoded data to decoders, some prefix strings can be predefined in encoders and decoders.

Session description protocol (SDP) is a text data that contains streaming media initialization parameters. An SDP data has the following properties

- It has mainly three description types which are session,time and media
- All descriptions are described one per line by a series of fields
- Description fields have two parts as static and dynamicstrings

The static parts are all known and do not change between SDP packets. Thus, an SDP package can be reconstructed with these dynamic parts and encoded representation of static parts.

Any text data with static and dynamic parts can be compressed by using LZ77 algorithm. A data compressed by LZ77 consists of *<length-distance>* pairs, uncompressed copies (references) of all pairs and uncompressed strings. However, this approach does not differentiate between the static and dynamic parts and apply the same algorithm for all the text data hence the redundancy caused by the static parts are not considered. In ESDiPi, we propose two enhancements for removing this redundancy

- All the static parts would be known to both coder and decoder parts hence there is no need to transfer those parts.
- The static parts are used as a prefix string and the pairs obtained use this prefix as reference. Therefore, the copies of static strings will be eliminated from the process.

Using known and static strings in prefix text makes the compressed data consist of only pairs and uncompressed data. Consider the example below with addition of proposed algorithm (LZ77+ prefix),

prefix → “ scream "
text → I scream you scream we all scream for ice cream.
LZ77 →I scream you<8,11> we all<8,17> for ice<5,14>.
LZ77+ Prefix →I<0,8>you<0,8>we all<0,8>for ice <2,5>.

Unlike LZ77, instead of *<length-distance>* pair *<offset-length>* pair may be used. Offset is the starting index of the substring in prefix and length is the length of it. Since many parts of the uncompressed data will be found in prefix string, using offset is more efficient than using backward distance. Although using prefix string is obviously useless for regular string data it is very efficient for text that have static parts.

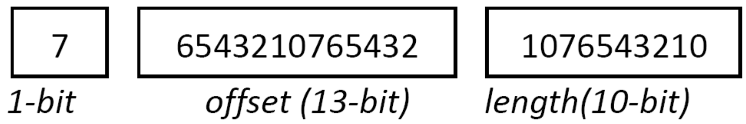

Fig. 1. Representation of length-distance pair for ESDiPi

An *<offset, length>* pair that is shown in Fig. 1 is represented by 3 bytes i.e. 24 bits with three parts. The parts can be expressed as follows:

1) 1 bit for data type which indicates whether it is anuncompressed byte or a beginning of offset, lengthpair
2) 13 bits for offset
3) 10 bits for length

Most significant bit distinguishes the byte sequencewhether it is the starting offset of the compressed data or it isan uncompressed data. Since any English letter in UTF8 orASCII encoding has byte code less than 128 compresseddata pair and letters can be distinguished according to thisbit.

Offset value is represented with the following 13 bitswhich can be up to 8192 thus uncompressed (prefix) textdata should not be more than 8192 bytes. For text datathat is longer than maximum size can be divided intosmall parts.

In addition, substring length can be up to 1024 bytes.Finding a substring in the prefix text compresses the substringto 3 bytes no matter how long it is. Since the order ofthe SDP lines can be reordered, lines of the uncompresseddata that can be reordered, if possible, to be found longer substringin prefix data.

Compressed data consists of concatenation of 3-bytepairs and uncompressed character bytes. There is no needto use any separator between bytes to distinguish the pairsand characters. Therefore, decoding the compressed datais straightforward. Decoding steps are given below:

1) Check the next byte whether its' most significant bitis 1 or not
2) If it is 1, take this byte with next two bytes and decodeit as shown above than go three bits ahead.
3) If it is not equal to 1, it means the byte is uncompressedjust write the character value and pass tonext byte.
4) Do first three steps until compressed data is consumed.

Information theory explains what cannot be compressed lossless or how much of a data can be compressed. According to that, it is not possible to compress random data that has high entropy or every symbol have equal probability. In an SDP package, there are some random crypto keys, fingerprints or passwords that cannot be compressed. Thus, this random information and the substrings that do not exist in prefix text are kept uncompressed.

4. Experimental Results

In this section, the proposed algorithm is tested using several length SDP packets from WebRTC communication sessions. The results are also compared to well-known Gzip, BZip, 7Zip and Zip algorithms.

4.1. Evaluation Metric

In lossless data compression, the data after decompression must be exactly same as the data before compression and there should not be any distortion in compressed data. The data compression ratio is defined as proportion of uncompressed and compressed size of the data. The compression ratio representation shows how much space will be used if one byte of data is decompressed. For instance, if 15MB of data becomes 3MB after compression, the compression ratio is shown as 5:1 that means that 1 unit of compressed data will take 5 units of space when it is decompressed.

Unlike, this usage space saving is used to calculate how much of the data is compressed. In this paper space saving that is defined in Definition 1 will be used as evaluation metric.

Definition 1. (Space Saving) assume C be a compression algorithm and $L(x)$ be the length function of data x. Space saving of data m is

$$R = 1 - \frac{L(m)}{L\big(C(m)\big)}$$

4.2. Comparison of the Algorithms

A typical session description protocol data takes approximately 3,000 bytes. The size can be changed according to the number of media (i.e., audio and video tracks) and number of network connection interfaces such as VPN, WiFi, 3G etc. The other parts such as session fields, some part of media fields and the order of the SDP lines are same for a system like WebRTC for iOS/ Android Native or WebRTC for Chrome. Many samples of session description protocols that have a size between 2,000 and 6,000 bytes are picked from these platforms and compressed sizes are compared.

Fig. 2 shows the compression amount of well-known compression algorithm Gzip (in red) and ESDiPi (in blue). These samples are also compressed with BZip, 7Zip and Zip algorithms and the average space saving values is given in Table 1.

In this experiment, a typical SDP data is used as a prefix string. By making some modifications on prefix, higher compression ration could be possible. For example, in video description the following part exists.

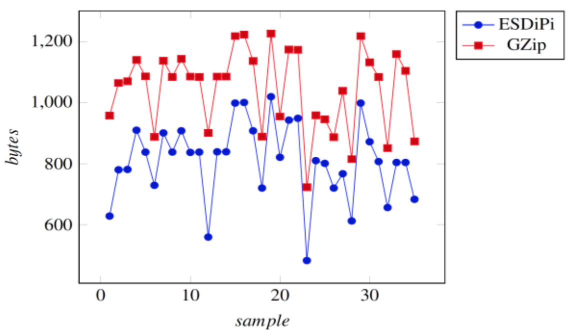

Fig. 2. Difference between GZip and ESDiPi.

Table 1. Space Saving Values of Algorithms

GZip	BZip	7Zip	Zip	ESDiPi
0.69	0.64	0.68	0.61	0.76

Note that in an SDP data of a specified platform (iOS, Android or Chrome), the above lines except direction (a=sendrecv) filed may be same. Thus, by changing direction value four different versions of the lines may be generated and added to prefix string. By this way, any SDP with any direction may be represented with only 3 bytes. Therefore, applying some tricks to prefix string may increase success ratio of ESDiPi on a specific platform.

5. Conclusion

In this work, a new compression algorithm is proposed for compressing SDP packets in the communications based on Lempel-Ziv coding using the specific SDP packet structure.. In mobile communications, Push Notificationmessage can be used to transmit SDP packets. The need to compress SDP packets risesfor two reasons

- The sizeof an SDP packet could be larger than a Push Notification message.
- There is user specific information in each Push Notification message which requires variable size storage.

There are many lossless compression algorithms, tocompress these kinds of data with high compression ratioi.e., Gzip, Bzip, Zip. Basically, they re-encode the text byusing a new coding system. However, these methods aregeneral purpose algorithms therefore any specific structure (correlation) of data is not considered. In ESDiPi, the static fields of theSDP packets are used as prefix text (known both to coderand decoder). By using prefix string for compressing thedata with LZ77 algorithm, we achieve higher compressionratios than well-known lossless compression methods.

The experimental results show that the proposed algorithmachieves higher space saving (or compression ratio) on several different SDP packets than universal compressionalgorithms such as DEFLATE.

5.1. Future Works

The proposed algorithm depends on LZ77 algorithm henceit only considers the repeating strings in the compressionstage. In addition since the prefix text data is predefinedfor compressor and decompressor, many of the redundantsubstrings will not exist in compressed data. Althoughhigh compression ratios are obtained with the proposed algorithm,the compression efficiency against other redundanciesin the data such as unequal entropy of symbols isvery low due to LZ77 dependency. To further increase thecompression ratio there are two possible directions,

- An additional entropy-coding step could be employed on ESDiPi compressed data.
- Uncompressed strings could be compressed using LZ77or another entropy coding algorithm.

In DEFLATE algorithm which is known to achieve high compression ratios apply Huffman coding in order to handle the unequal entropy of LZ77 compressed symbols. With a similar strategy, Huffman coding would be used to code ESDiPi compressed symbols to achieve even higher compression ratios. Similarly, uncompressed parts of SDP packets would be compressed using the same algorithm.

References

[1] Handley, M., Jacobson,V., & Perkins, C. (July 2006). SDP: Session description protocol. *Network Working Group*. Retrieved June 15, 2015, from https://tools.ietf.org/html/rfc4566

[2] Loreto, S., Saint-Andre, P., Salsano, S., Wilkins, G. (April 2011). Known issues and best practices for the use of long polling and streaming in bidirectional. *Internet Engineering Task Force*. Retrieved June 15, 2015, from https://tools.ietf.org/html/rfc6202

[3] Deutsch, P. (May 1996). Deflate compressed data format specification version 1.3. *Network Working Group*. Retrieved June 15, 2015, from https://www.ietf.org/rfc/rfc1951.txt

[4] Castillo, I., Villegas, J., & Pascual, V. (January 2014). The websocket protocol as a transport for the session initiation protocol (sip). *Internet Engineering Task Force*. Retrieved June 15, 2015, from https://tools.ietf.org/html/rfc7118

[5] Fette, I. (December 2011). The websocket protocol. *Internet Engineering Task Force*. Retrieved June 15, 2015, from https://tools.ietf.org/html/rfc6455

[6] Atalay, A. (June 2015). Signaling mechanism design for WebRTC. *ISITES 2015*.

[7] Apple Inc. *Apple Push Notification Service.*

[8] Google. *Google Cloud Messaging*.

[9] Huffman, D. (1952). A method for the construction of minimum redundancy codes. *Proceedings of the Institute of Radio Engineers*: *Vol. 40* (pp. 1098–1101).

[10] Sharma, M. (2010). Compression using Huffman coding. *International Journal of Computer Science and Network Security, 10,* 133–141.

[11] Ziv, J., Lempel, A. (1997). A universal algorithm for sequential data compression. *IEEE Transactions on Information Theory*, *23*, 337–343.

[12] Brian, W. (2005). *Handbook of Image Quality: Characterization and Prediction*. CRC Press.

[13] Deutsch, P. (May 1996). Gzip file format specification version 4.3. *Network Working Group.* Retrieved June 15, 2015, from https://www.ietf.org/rfc/rfc1952.txt

[14] Wikipedia. (2015). Jpeg compression. Retrieved June 15, 2015, from https://en.wikipedia.org/wiki/JPEG

Adem Atalay received his B.Sc. degree of mathematics in 2006 from Middle East Technical University, Ankara, Turkey and the M.Sc. degree in mathematics in 2011 from Kocaeli University, Kocaeli, Turkey. He is studying for his Ph.D. degree in computer sciences at Yıldız Technical University, Istanbul, Turkey.

He worked in cryptography and security field from 2007 to 2011 as a cryptography design and analysis researcher in National Electronic and Cryptography Research Institute. Afterwards, he founded a mobile software development company and managed the company from 2011 to 2014. Since 2014, he has been working as a mobile software architect at Netaş Telecommunications R&D in Turkey.

Doğaç Başaran received his B.Sc., M.Sc. and Ph.D. degrees in Electrical and Electronics Engineering Department from Boğaziçi University, İstanbul, Turkey, in 2002, 2005 and 2015 respectively. He worked as a research assistant at TAM Project that is supported by the Turkish State Planning Organization (DPT). He is currently working as a research engineer at Netaş Telecommunications R&D company where he is working on multimedia communications. His research interests are in audio (Music-Speech) signal processing, statistical signal processing, Bayesian and approximate inference, machine learning, blind source separation, multiple audio sequence alignment, multimedia communications, 4G-LTE and 5G networks, internet of things.