Proceedings of the 7th International Conference on Applied Informatics
Eger, Hungary, January 28–31, 2007. Vol. 2. pp. 317–325.

SIP compression in mobile device environment*

Zalán Szűgyi, Zoltán Porkoláb

Eötvös Loránd Tudományegyetem
e-mail: gsd@inf.elte.hu, lupin@ludens.elte.hu

Abstract

Protocol (SIP) to establish, modify, and close sessions. The SIP messages are text based and they have redundancy. Since in wireless environments the bandwidth is limited, we need to decrease traffic during communication. One solution is Signalling Compression (SigComp) which offers a robust, lossless compression of SIP messages in a standardized way.

In this article we overview the possibilities to improve effeciency of SigComp. The standard defines the major functionality, but there are several cases where the implementator can make decisions. The compressor modul is the one where the standard give the most freedom to improve SigComp. We have chosen the LZSS algorithm and looked for a good match function. We examined the detailed possibilities and tuned different parameters regarding the restricted resources of a mobile phone environment. Our result: in common usage the general algorithms were better, but using in SIP environment our solution produced better efficiency.

Keywords: SIP, SigComp, Compression, LZSS

MSC: 68P30 Coding and information theory (compression)

1. Introduction

Nowadays the mobile phones has a rapid development. Some years ago they have been used to make phone calls or send short messages only, but today in addition we can use them to take photos, listen music, browse the Internet etc. The new ones have operation system, so their users can install any kind of third party software. As we can see, mobile phones are not just phones, they are small computers. Now we can browse the Internet via WAP or connect to our Internet provider via GPRS, and we can make phone calls from PC to mobile phone by Skype or any other application using VoIP technology. However special gateways need to establish these interactions. In the near future this will be much easier. The

*Supported by Stiftung Aktion Österreich-Ungarn, Pr.N.: 66öu2.

3rd Generation Partnership Project (3GPP) (see [1]) chose the Internet Protocol to the base of Universal Mobile Telecommunication System (UMTS). This means each mobile phone will have its own IP address, and will be directly reachable by every other member of the Internet.

There are several applications on the Internet for different type of real time communication, such as voice call or video conference. These applications use session base connections and they have their own well designed and well tested protocols to manage the communication. When the new generation mobile system will be introduced it should be expedient to apply these protocols to mobile devices instead of developing new ones. During these implementations some new problems must be solved, which do not exist in the original way. For example one of these problems is: We are sitting in a car and talking via mobile phone. During the talk, we leave the area of one mobile cell, and enter to another one. The used communication protocol has to find the new location of our mobile device without disconnecting.

The solution for these problems is the Session Initiation Protocol (SIP) (see [2]), which is not a standalone protocol to replace the others, but extends their functionality. There is another problem using Session Initiation Protocol. The messages of SIP are text based, so the length of these messages is quite big. The bottleneck of mobile communication is a bandwidth, so it is necessary to reduce the size of these messages as much as possible.

The Signalling Compression (SigComp, see [3, 4]) subsystem is responsible to compress SIP messages. Every message need to be compressed, so it is very important to do the compression fast and efficient. This article introduces one way to implement an efficient SigComp subsystem.

The development is made in Nokia Hungary Kft, and our solutions are used in business applications of this company.

In the second chapter the Session Initiation Protocol is introduced. In the third chapter the five components of the Signalling Compression are described. In the fourth one there are some common compression algorithms and methods which are proper to use in mobile environment. In the fifth chapter you can find our solution, and in the sixth one we justify our decisions.

2. Session Initiation Protocol

The Session Initiation Protocol (see [2]) is not the only standalone protocol, which will replace the other session protocols in the future, but improves their functionality. Let us look how does the session based communication work. To establish sessions the proper protocol must exist (e.g. VoIP for voice communication). First the other endpoint need to be localized. Then we need to make sure, the other endpoint is able to accept our request, and the proper protocol exists in that side too (e.g. the old mobile phones cannot receive picture messages). These tasks are made by SIP. When the connection is established, the SIP goes to the background and give the control to the protocol, which suits to the chosen type of communication. When the communication finishes the SIP closes the connection. The whole lifetime of a session based communication on mobile device can be seen on the first figure, where Alice makes a phone call to Bob.

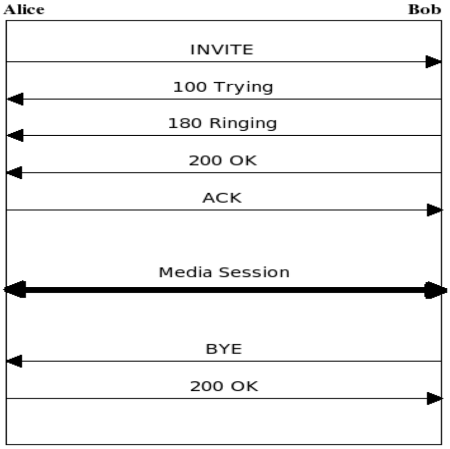


Figure 1: Sip message flow

The five main task of SIP

- User location: determination of the end system to be used for communication
- User availability: determination of the willingness of the called party to engage in communications
- User capabilities: determination of the media and media parameters to be used
- Session setup: “ringing”, establishment of session parameters at both called and calling party
- Session management: including transfer and termination of sessions, modifying session parameters
- and invoking services

3. Signalling Compression

As we can see in the previous chapter, several SIP messages are exchanged to establish sessions. This messages are text based. In mobile environment the bandwidth is the bottleneck, so it is very important to reduce the length of SIP messages. The Signalling Compression subsystem (see [3, 4]) is responsible to compress these messages. The SigComp lays between local application and transport layer. It means the application is not necessary to know whether the message is sent or received compressed or uncompressed. On the second figure the overview of the SigComp can be seen.

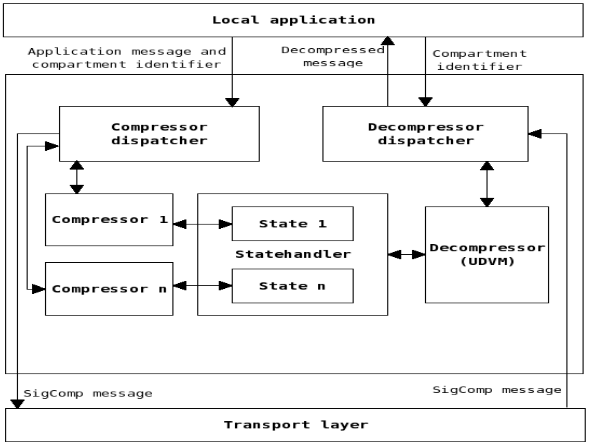


Figure 2: Overview of SigComp

There are five main modules of SigComp.

Compressor / Decompressor dispatcher

Just an interface. Provides some methods for the application and the transport layer to send messages compressed, or uncompress received messages.

Compressor

This module does the compression. A mobile switch center can maintain several connections at the same time. Each connection has its own compressor module. The compression algorithm can be different in every module.

Decompressor

This module does the decompression. There is an Universal Decompressor Virtual Machine (UDVM) in the decompressor side, which is similar to JVM but its instructions designed to decompress messages efficiently. The first message must contain the byte code of decompression algorithm to UDVM. That is why we can use different compression algorithms.

Statehandler

This module can store information about compression and decompression to improve the efficiency of the next message's compression. For example we use a dictionary based compression algorithm. At the beginning the dictionary can be filled with previously saved information about frequently used keywords. In this way the compression will be better then the one with an algorithm started with an empty dictionary. The UDVM byte code is stored in the Statehandler also.

4. Compression

There are several algorithms, which can compress SIP messages efficiently. These are: Rice, SubExponential, Huffman, LZ77, LZSS, Deflate etc. Based on a study from Szeged University (see [5]) about Signalling Compression, we chose LZSS algorithm (see [6, 7]), because the decompression time is much less then with the others, and the compression ratio is almost the same.

The LZSS algorithm is a dictionary based compression algorithm. The compression algorithm has three main steps:

1. Find the longest prefix of the message in the dictionary.
2. Write the (p, n) pair, where p is the position of the prefix in the dictionary, and the n is the length of it.
3. Write the prefix to the end of the dictionary.

In the zero step we fill up the dictionary with previously saved information about the other messages.

We implemented the dictionary by circular buffer. It means that, when we write the prefix into the dictionary and reach the end of it, then we continue the writing at the beginning of the dictionary, we rewrite the characters there.

The decompression algorithm's main steps:

1. Read the next (p, n) pair from input.
2. Jump to the p_felso index_th. position of the dictionary and write out n characters.
3. Write the string coded by (p, n) to the end of the dictionary.

In zero step we must fill up the dictionary the same way we did it at the compression side.

5. The match function

The match function is responsible to find the longest match in the dictionary. This is the bottleneck of the algorithm, so it must be implemented efficiently. There are several ways to implement the match function (see [8]).

Knuth-Morris-Pratt

His algorithm is an improved linear search. It can be faster then general linear search by analyzing the pattern. This way is efficient when the pattern has big repetition. Our patterns are mainly English words, so this method is not good for us.

Lists

In this way we assign a linked list to each character of the alphabet. In the lists we store the positions of the proper characters appearing in the dictionary. To find the longest prefix it is enough to examine the dictionary at those positions, which are staying in the list belongs to the starter character of the prefix. We can improve this way to assign lists to two or three tuples of characters instead of single ones. This way is quite good for us, because there is no big repetition in the English texts, so the size of these lists are low. The disadvantage of this method is the huge memory consumption, because we need to store at least one pointer for every list, even they are empty.

Hash

Improvement of lists. We map the n-tuples of characters to the set of keys which has significantly less elements. Then we assign lists to the keys only. This is good for us because the most n-tuple's list are usually empty, so the key collusion is rare. In this way we can save a lot of memory.

6. One solution

The mobile switch centre can maintain hundreds of connections at same time, sending and receiving SIP messages. Every outgoing message must be compressed and every incoming message must be decompressed. Fast compression of SIP messages is very important. The bottleneck of the LZSS compression is the match function, so we tried to improve that.

Our solution is based on hash. Because the dictionary is implemented by a circular buffer, it can happen that an old part of the dictionary is rewritten by a new one. In that way the characters are changed in the current position. So this position value must be removed from the list belongs to the old tuple and inserted to the new one.

Implementing our data structures of std::list (see [9]) makes two problems.

- Replace a position value from one list to another implies a node allocation and deallocation, which are expensive.
- We can find the position value to remove only in linear time.

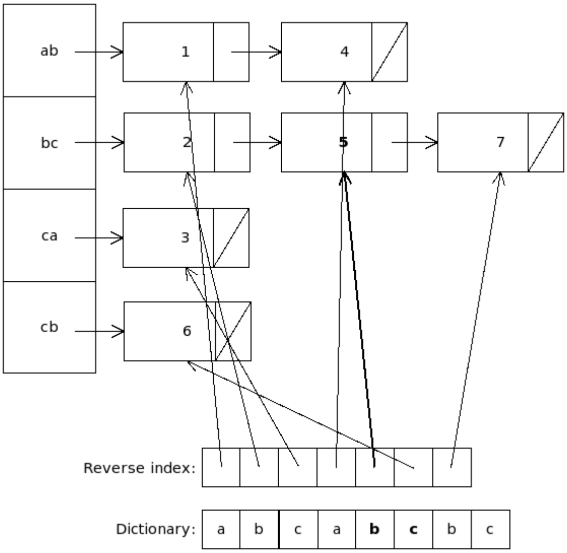


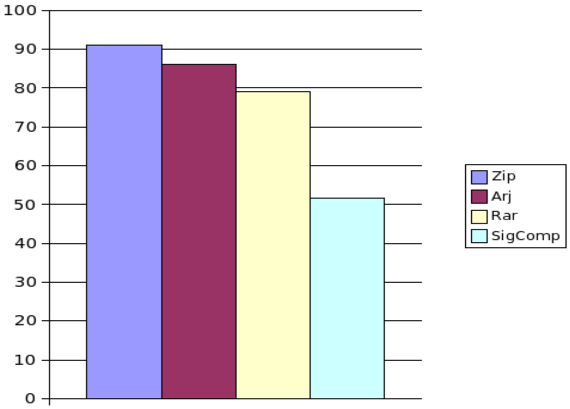
Figure 3: Solution with reverse index vector

Our solution eliminates these problems. We introduced a new data structure. It can be seen in the third figure.

On the left hand side vector we store pointers to the lists belonging to the keys mapped to the pair of characters. These character pairs can be seen in the vector. The list nodes contain the position values. In the Reverse index vector we store a pointer to the node whose value is the same as its index. (Let suppose we start to index the vector by 1.)

Let us see how it works. For example the message we want to compress starts with “bc”. Now we calculate the key value of “bc” string which is two. After that we get the second element of the vector, which contains a pointer to the proper list. It is enough to find the longest prefix in the dictionary at 2nd, 5th and 7th position because only those parts start with “bc”. After finding the longest prefix we need to attach it to the dictionary. Because of circular buffer some parts of the dictionary can be rewritten. Let suppose the ’b’ character on the fifth position changes. Now we must remove the position 5 from the list belonging to “bc” and insert it to the proper list (it depends on the new character). We can find the node contains 5 in constant time, because it is stored in Reverse index vector. Instead of deallocating the old node and allocating a new one it is enough to unlink it from the old list and link it to the new one. The unlink can be done in constant time because we use two direction linked lists. The lists can be unordered because finding the longest prefix does not depend on the order of the positions and with the help of Reverse index we can find every node in constant time. We can link the node to the front of the list, so it can be done also in constant time. In this way we can compress messages three times faster then we use brute force solution for match function.

Eventually we compared our solution with some well known common compressor applications such as Zip (see [9]), Arj (see [10]), Rar (see [11]). Generally these algorithms were better. But the SIP messages are relatively small. Their size is about one or two kilobyte. There are plenty of SIP messages travelling from client to server and vica versa during communication, and we have to compress these message separately. In this way, when the text that we compress is small, our solution is almost two times better then the general compressor applications. We compressed many SIP messages with Zip, Arj, Rar and our SigComp solution, and the average compression rates can be seen in the following picture.



7. Summary

The bandwidth is the bottleneck of mobile communication, so the compression of messages is very important in the next generation of telecommunication. The efficient compression has two meanings. One is to reach good compression ratio, and the second is to do it fast. Our solution suits for both requirements. We can compress the messages to half of their original size, and we can do it about three times faster then useing brute force algorithm for match function. The Nokia Hungary Kft. will apply our solution to its future business application. The 3G technology is ready to be introduced all over the telecommunication world, and soon it will replace the currently used GSM (see [1]) system. There are some countries such as France and Italy where an earlier version of this technology is already available.

References

[1] www.3gpp.org

[2] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., Schooler, E., Session Initiation Protocol, RFC 3261, (2002).

[3] Price, R., Bormann, C., Christoffersson, J., Hannu, H., Liu, Z., Rosenberg, J., Signaling Compression1 RFC 3320, (2003).

[4] Hannu, H., Christoffersson, J., Forsgren, S., Leung, K.-C., Liu, Z., Price, R., Signaling Compression (SigComp) - Extended Operations, RFC 3321, (2003).

[5] Fridrich, M., Bohus, M., Sógor, L., Sógor, Z., Bilicki, V., Galambos, Zs., Szilágyi, T., Notaisz, K., Siket, P., Siket, I., Sey, G., Márton, Zs., Tóth, G., István, R., Study Report of the Project: Signalling Compression Protocoll Development (2002).

[6] Cormen, T. H., Leiserson, C. E., Rivest, R. L., Algoritmusok, (2001).

[7] Sándor, S., Programozási Módszertan II: Algoritmusok.

[8] Bell, T., Kulp, D., Longest-match String Searching for Ziv-Lempel Compression, (1993).

[9] http://www.cppreference.com/cpplist/index.html

[10] http://en.wikipedia.org/wiki/ZIP_(file_format)

[11] http://en.wikipedia.org/wiki/Arj

[12] http://en.wikipedia.org/wiki/Rar

Zalán Szűgyi, Zoltán Porkoláb
Eötvös Loránd Tudományegyetem
Dep. of Programming Languages and Compilers
H-1117 Budapest, Pázmány Péter sétány 1/C.
Hungary