MINIATURE SIP FOR EMBEDDED SYSTEMS

Leonardo Maccari Rufino and Antônio Augusto Fröhlich
Federal University of Santa Catarina
Laboratory for Software and Hardware Integration
P.O.Box 476, 88040900 - Florianópolis - SC - Brazil
Email: {leonardo,guto}@lisha.ufsc.br

ABSTRACT

Session Initiation Protocol (SIP) is an application layer protocol responsible for creating, modifying, and terminating a session. It is used to establish calls through networks via IP protocol. Nowadays, SIP has been used by many embedded systems, however, due to its large size, some constrained systems cannot use it. This paper aims at adapting the protocol to be used in resource-constrained embedded systems, like household appliances and low cost surveillance cameras. In order to achieve this objective, many requests and header fields were suppressed, without compromising its functionalities, resulting in a reduced SIP. The reduced version was implemented in an embedded operating system, reaching a final system image of 116 Kbytes.

KEYWORDS

SIP, embedded systems, resource constraints.

1. INTRODUCTION

SIP (Session Initiation Protocol) is an application layer protocol responsible for creating, modifying, and terminating a session (e.g. multimedia communication session such as voice and video calls) (Rosenberg et al. 2002). This protocol is used to establish calls through networks via IP protocol. Due to its characteristics, the SIP has been widely used in embedded systems, like cell phones, PDAs, and web cameras (Lakay and Agbinya 2005). However, the protocol lacks on performance (e.g. size of code) and cannot be used in resource-constrained embedded systems (e.g. household appliances and low cost surveillance cameras).

Some embedded systems do not have one person controlling it. These kinds of systems are the target of this work. An example is a security camera, where a session can be started either by a user making a call via a SIP phone for the address of the camera or by the camera itself, when motion is detected.

This paper proposes an adaption of the SIP protocol in order to enable its utilization in resource-constrained embedded systems. The proposed implementation of the protocol does not use some features, and requests and headers fields present in a full SIP version. Nevertheless, the modifications have not compromised the protocol functionality. The miniature SIP version was implemented using the Embedded Parallel Operating System (EPOS) (Fröhlich 2001), with a total of 116Kb of code and data.

The rest of this paper is organized as follow: Section 2 presents the related work. The proposed SIP implementation is described in Section 3. Section 4 shows the results. Finally, Section 5 concludes the paper.

2. RELATED WORK

Real-Time Visitor Communication Service (RVCS) is an architecture based on the home gateway system and SIP protocol, in which the resident of a home can monitor visitors in front of his/her door using devices connected to the Internet, such as PCs and PDAs, supporting user mobility (Oh et al. 2006).

SIP Context-Aware Gateway (SCAG) represents an intelligent gateway deployed between the home network and the Internet, allowing the homeowner to use his/her preferred SIP devices to communicate and post their context information for household appliances. It makes possible to smart home applications offer services that can be adapted to user's dynamic situations (e.g. user location) and gives special treatment to their preferences (Cheng et al. 2006).

In addition, a design of a ubiquitous services system, which provides mobility and uniform interface to the user, using SIP, was proposed by Kwak (Kwak 2007). When a user wants to enter a SmartSpace (scenario which consists of intelligent services accessible to mobile users via handheld devices connected over wireless links for short distance), it registers its device in to a manager. Thus, the portable device receives the updated list of devices that exists in the SmartSpace and, after selecting the desired service and the destination device, the user is able to connect to it via a uniform interface based on touch event operations. Then, the requested service can be offered to the user.

The ability of processing SIP messages quickly is critical to the performance of consumer electronic devices in home network. Taking into account that the entire message parsing is the performance bottleneck of SIP servers, Demand-Driven Parsing Method (DPM) only parses the message types defined in a XML document, ignoring the others, thus enabling to save the processing time (Liao et al. 2009).

3. PROPOSED IMPLEMENTATION FOR THE SIP PROTOCOL

In order to obtain a miniature version of the SIP protocol able to run in a resource-constrained embedded system, without compromising its functionalities, only a subset of request and header fields were used. In the next subsections, we present the request and header fields that were implemented and discuss why some fields were not needed.

3.1 Requests

Among all the request messages that the SIP has, the implemented version is restricted to only three: *INVITE*, *ACK*, and *BYE*. Such messages represent the methods required for a session to be started and completed. Other requests such as those present in the original standard *CANCEL*, *OPTIONS*, and *REGISTER* (Rosenberg et al. 2002) and those located in separate RFCs, such as *REFER*, *SUBSCRIBE*, *NOTIFY*, *MESSAGE*, *UPDATE*, *INFO*, and *PRACK* (Roach 2002), have not been implemented. These kinds of requests are extensions of standard and therefore are optional. Consequently, they are not used in the SIP adaptation proposed in this paper.

Related to the SIP RFC requests, *CANCEL* is used to cancel a request previously sent by a UAC (User Agent Client). This message is not used because when an invitation to start the session comes from the embedded system, there is no need to cancel the attempt call. But when the attempt is made by the opposite side, as the system has no user, there is no need to send provisional responses (e.g. ringing) and the connection should be accepted or rejected instantly. Then, the *CANCEL* method can be suppressed without impact on functionality, because it has no effect on a request to which a UAS (User Agent Server) has already sent a final response, being applicable in situations where the server side can take a long time to respond to a request.

The *OPTIONS* method is used to query a UA (User Agent) or proxy server about their capabilities without the need to place a call with the other party. For example, before a UAC sends an *INVITE* message with the *Require* header field, it can send an *OPTIONS* to make sure that the UAS supports the required fields. This sequence of two messages, *OPTIONS* and *INVITE*, has the same effect of sending *INVITE* twice. In this situation, the first invitation requires a few options. However, if the called party does not support the required extensions, the standard procedure is to reject the request, sending a header field explaining the reason. Therefore, a second *INVITE* can be generated with only the options allowed by the UAS. Wherefore, this request can be removed without any problem.

REGISTER is a message used by a UA to notify a SIP network of its *Contact* URI and a URI which should have requests routed to this contact. This work suppressed this method because each UA has a fixed IP address. Then, the message exchanges are sent directly to the IPs of end-points desired, without the use of servers (proxy, redirect, and registrar).

3.2 Header Fields

The SIP protocol has 44 header fields described in its standard. Among these, only some have been used in this work: *Allow*, *Call-ID*, *Contact*, *Content-Disposition*, *Content-Length*, *Content-Type*, *CSeq*, *From*, *Max-Forwards*, *Record-Route*, *Require*, *Route*, *To*, *Unsupported*, and *Via*.

Via, *From*, *To*, *CSeq*, *Call-ID*, *Max-Forwards*, and *Contact* are needed in the most of the SIP messages. *Content-Disposition*, *Content-Length*, and *Content-Type* identify the packets contained in the SIP message body. *Record-Route* and *Route* provide routing information. *Allow*, *Require*, and *Unsupported* report requests and header fields that may or may not be used.

All 29 remaining header fields were excluded. A major reason for the exclusion is that a portion of them only convey information about a user to another, however there is no need to exchange these data. Some headers that are included in this context are: *Accept-Language*, *Alert-Info*, *Call-Info*, *Content-Language*, *Date*, *Error-Info*, *Organization*, *Priority*, *Retry-After*, *Server*, *Subject*, *Timestamp*, *User-Agent*, and *Warning*.

Accept, *Accept-Encoding*, *Content-Encoding*, *MIME-Version*, *Proxy-Require*, and *Supported* were deleted from the miniature version because there is no available encoding for them, in other words, their fields would be empty if necessary send them. For instance, *Content-Encoding* indicates what additional content coding (e.g. gzip) have been applied to the message body.

This study does not require authentication schemes because it is designed for confined systems, not connected to the Internet. Consequently, some headers were excluded, as: *Authentication-Info*, *Authorization*, *Proxy-Authenticate*, *Proxy-Authorization*, and *WWW-Authenticate*. The remaining header fields were removed for their own reasons, for instance, *Expires* indicates the time that the message contents is valid, since after a response this field no longer has meaning, and as the responses are sent immediately on receiving a request, without having to wait for user actions, this header does not apply. *In-Reply-To* enumerates the Call-IDs that a call references or returns, *Min-Expires* transmits the minimum expiration time supported by a registrar server in response to a request *REGISTER*, as this work does not use this method, this header also becomes unnecessary. Finally, *Reply-To* specifies a URI that should be used in response to a request, however, as we are sending messages from fixed IP addresses, there is no need for a system has more than one URI identifying it.

The designed version has also the four state machines needed by the SIP protocol: *INVITE* client transaction, non-*INVITE* client transaction, *INVITE* server transaction, and non-*INVITE* server transaction. The miniature SIP version was implemented in the Embedded Parallel Operating System (EPOS) (Fröhlich 2001). EPOS was chosen because it has a small footprint and a complete communication stack, including UDP (used in the transport layer) and support for sensor networks (Fröhlich and Wanner 2008), without user interaction.

Hence, with all these changes made in the original version of the SIP protocol, there is a great economy of memory, achieving a final system image with a small size. Then, this proposed implementation can be used even in the smallest and simplest embedded systems present today.

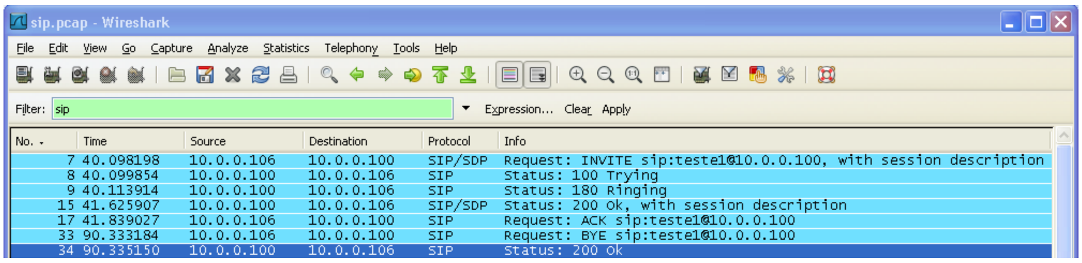
4. RESULTS

To evaluate the applicability of the proposed SIP implementation for embedded systems, we have measured the generated library. Table 1 shows the size of the version created, comparing to other well-known SIP implementations, like oSIP library (Moizard 2002) and the part referring to SIP in Asterisk (Digium 1999). All codes were compiled for the IA32 architecture, using the GNU gcc compiler version 4.0.2 and measured with the ia32-size tool, version 2.16.1. The results show that, with all the reductions made, it was obtained the final library size of 87,450 bytes, almost three times smaller than oSIP library and eight times smaller than Asterisk. The final image size, with EPOS and SIP version, reached 118,532 (116 Kbytes), much less than the others, even without considering the several megabytes of operating system overhead.

Table 1: Size of SIP implementations in bytes.

Section	SIP	libosip	Asterisk
.text	81,074	232,914	609,678
.data	6,360	3,220	7,912
.bss	16	1,172	67,972
TOTAL	87,450	237,306	685,562

In order to demonstrate the functionality, a communication test between two machines was performed. On one side, the EPOS operating system with the SIP implemented, running on VirtualBox virtual machine. On the other side, a machine with a SIP phone ready to make and receive calls. It was tested the startup and shutdown of sessions across the two machines. Figure 1 shows the test output in the format of messages flow through the graphical interface of the Wireshark program. The IP 10.0.0.106 represents the virtual machine with EPOS, while 10.0.0.100 a machine running Windows XP operating system and X-Lite SIP phone.


sip.pcap - Wireshark
File Edit View Go Capture Analyze Statistics Telephony Tools Help
Filter: sip
Expression... Clear Apply

```
No. -  Time       Source      Destination  Protocol  Info
     7 40.098198  10.0.0.106  10.0.0.100   SIP/SDP   Request: INVITE sip:teste1@10.0.0.100, with session description
     8 40.099854  10.0.0.100  10.0.0.106   SIP       Status: 100 Trying
     9 40.113914  10.0.0.100  10.0.0.106   SIP       Status: 180 Ringing
    15 41.625907  10.0.0.100  10.0.0.106   SIP/SDP   Status: 200 Ok, with session description
    17 41.839027  10.0.0.106  10.0.0.100   SIP       Request: ACK sip:teste1@10.0.0.100
    33 90.333184  10.0.0.106  10.0.0.100   SIP       Request: BYE sip:teste1@10.0.0.100
    34 90.335150  10.0.0.100  10.0.0.106   SIP       Status: 200 Ok
```

Figure 1: Initialize and finalize a session with SIP in Wireshark.

5. CONCLUSION

This paper proposed adaptations to the SIP protocol to make it feasible in resource-constrained embedded systems. By using only a subset of the total group of methods present in the standard, as well as a subset of header and request fields, it was possible to generate a small version of the SIP protocol without compromising its functionalities.

The results have shown that the proposed SIP miniature version consumes less than 120Kb of code and data. Through a test scenario, we also demonstrated that it is able to communicate to any SIP phone, starting and ending a session.

As future work, we intend to implement the developed version in a real embedded system and to study other necessary protocols for communication, such as RTP.

REFERENCES

Cheng, B. et al, 2006. Context-Aware Gateway for Ubiquitous SIP-Based Services in Smart Homes. *International Conference on Hybrid Information Technology*, Vol. 2, November 2006, pp. 374-381.

Digium. 1999. *Asterisk – The Open Source Telephony Projects*. [Online] Available at: http://www.asterisk.org/ [Accessed 15 July 2010].

Fröhlich, A. A., 2001. Application-Oriented Operating Systems. 1st ed. GMD – Forschungszentrum Informationstechnik, Sankt Augustin, Germany. ISBN: 3-88457-400-0.

Fröhlich, A. A. and Wanner, L. F., 2008. Operating System Support for Wireless Sensor Networks. *Journal of Computer Science*, Vol. 4. No. 4, pp. 272-281.

Kwak, J, 2007. Ubiquitous Services System Based on SIP. *IEEE Transactions on Consumer Electronics*, Vol. 53, No. 3, August 2007, pp. 938-944.

Lakay, E. T.; Agbinya, J. I., 2005. SIP-based content development for wireless mobile devices. 1st International Conference on Computers, Communications, & Signal Processing with Special Track on Biomedical Engineering, November 2005, pp. 130 - 134.

Liao, J. et al, 2009. A demand-driven parsing method for SIP offload in home network. *IEEE Transactions on Consumer Electronics*, Vol. 55, No. 3, August 2009, pp. 1308-1314.

Moizard, A. 2002. *The GNU oSIP library*. [Online] (Updated 03 October 2008) Available at: http://www.gnu.org/software/osip/ [Accessed 13 July 2010].

Oh, Y. et al, 2006. Design of a SIP-based Real-time Visitor Communication and Door Control Architecture using a Home Gateway. *IEEE Transactions on Consumer Electronics*, Vol. 52, No. 4, November 2006, pp. 1256-1260.

Roach, A. B., 2002. Session Initiation Protocol (SIP)-Specific Event Notification. RFC 3265.

Rosenberg, J. et al, 2002. SIP: Session Initiation Protocol. RFC 3261.