Building Web-base SIP Analyzer with Ajax Approach

Quincy Wu and Yan-Hsiang Wang
Department of Computer Science and Information Engineering
National Chi Nan University
No. 1, University Road, Puli, Nantou 545, Taiwan
{solomon,s94321517}@ncnu.edu.tw

Abstract

Web applications are generally less interactive than desktop applications. Due to the simple request-response model between web browsers and web servers, users usually experience frequent waiting during a session whenever the web applications need to get data from the server. After a browser sends out a request and await the response to return from the server, no further requests can be taken during this waiting time.

In this paper we propose integrating the Ajax approach to the design of a web-base analyzer for Session Initiation Protocol (SIP), which is a popular protocol in IP telephony industry. By getting data asynchronously from the server, this web-base analyzer can monitor the SIP messages on the server in real-time without reloading.

Keywords: Ajax, libpcap, packet analyzer, SIP

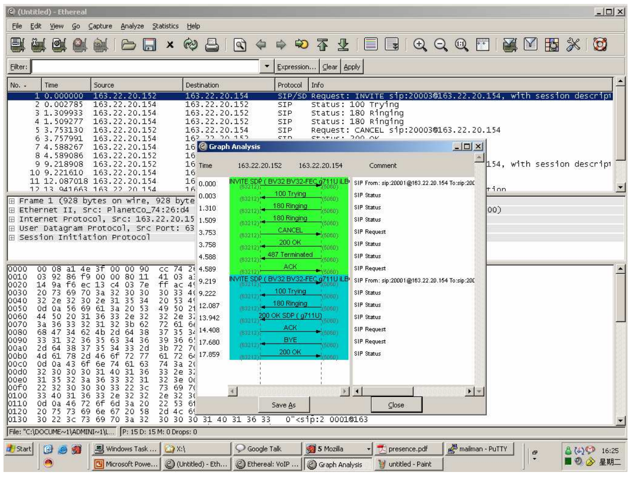
1 Introduction

Internet multimedia applications like Internet telephone calls, multimedia streaming, and multimedia conferences are growing in popularity. Along with H.323 and others, Session Initiation Protocol (SIP) is widely used as signaling protocol for Voice over IP. It is an application layer signaling protocol for Internet multimedia session establishment, modification, and termination [9]. In November 2000, SIP was accepted by the Third Generation Partnership Project (3GPP) as a signaling protocol and permanent element of the IP Multimedia Subsystem (IMS) architecture.

Because of the flexibility and rich features of SIP, more and more Internet telephony service providers (ITSP) adopt it as the protocol in delivering their services. A growing number of SIP-based services and products are now becoming available in the market, including SIP servers, SIP gateways, SIP firewalls and network address translators (NATs), and SIP phones (which are known as ”SIP user agents” in its terminology) [8]. Common appearance of SIP user agents includes IP phones, USB phones, video phones, softphones, and WiFi phones. When programmers need to develop the SIP system on these user agents, generally it will be very helpful to have a tool to capture the SIP messages for real-time or postmortem analysis. For softphone programmers, this is not a difficult task. As shown in Figure 1, on desktop or laptop computers, it is easy to find user-friendly packet analyzers such as Ethereal [2], which runs on Linux, FreeBSD, and Microsoft Windows 2000/XP/2003, to provide rich functions in capturing network packets and performing further analyses. On the contrary, for embedded devices such as IP phones and WiFi phones, generally the limited computing power prohibits them from running a sophisticated utility like Ethereal. Even worse, these devices generally have small LCD displays with low resolution, or LED displays which can only show two or three text lines. It is certainly unfavorable to show the packet analysis with a small display on this kind of device.

Since SIP user agents will communicate via SIP servers, one of the common measures taken by developers is to run the packet analyzer on SIP servers. As long as the packet analyzer supports a remote displaying mechanism such as X Window, the developer can bring a WiFi phone and his/her laptop to a distant site, run the packet analyzer on the server, and display the output to the laptop in real time. This allows the developer to test the WiFi phone in a location that is distant from the SIP server, while keeping capable of monitoring the SIP messages between the WiFi phone and the SIP server. However, for a tool like Ethereal to be able to capture the network packets sent and received on the SIP server, generally the super-user privilege is required to run this program on the server. For an ITSP that is running an interoperability test with a vendor of a new model of WiFi phone, generally this imposes concerns for possible compromise of security and confidentiality.

Some web-base packet analyzers were developed to address this issue, such as WIST [1] and Distributed SIP Analyzer [7]. In WIST, a Unix tool ”ngrep” is running on the server to capture SIP messages and save them in a log file.



```
No. -  Time         Source          Destination     Protocol Info
     1 0.000000     163.22.20.152   163.22.20.154   SIP/SD   Request: INVITE sip:20003@163.22.20.154, with session descript
     2 0.002785     163.22.20.154   163.22.20.152   SIP      Status: 100 Trying
     3 1.309933     163.22.20.154   163.22.20.152   SIP      Status: 180 Ringing
     4 1.509277     163.22.20.154   163.22.20.152   SIP      Status: 180 Ringing
     5 3.753130     163.22.20.152   163.22.20.154   SIP      Request: CANCEL sip:20003@163.22.20.154
     6 3.757991     163.22.20.154   16
     7 4.588267     163.22.20.154   16
     8 4.589086     163.22.20.152   16
     9 9.218908     163.22.20.152   16                                          154, with session descript
    10 9.221610     163.22.20.154   16
    11 12.087018    163.22.20.154   16

⊞ Frame 1 (928 bytes on wire, 928 byte
⊞ Ethernet II, Src: PlanetCo_74:26:d4                                           00)
⊞ Internet Protocol, Src: 163.22.20.15
⊞ User Datagram Protocol, Src Port: 63
⊞ Session Initiation Protocol

0000  00 08 a1 4e 3f 00 00 90  cc 74 2
0010  03 92 86 f9 00 00 80 11  41 03 a
0020  14 9a f6 ec 13 c4 03 7e  ff ac 4
0030  20 73 69 70 3a 32 30 30  30 33 4
0040  32 2e 32 30 2e 31 35 34  20 53 4
0050  0d 0a 56 69 61 3a 20 53  49 50 2
0060  44 50 20 31 36 33 2e 32  32 2e 3
0070  3a 36 33 32 31 32 3b 62  72 61 6
0080  68 47 34 62 4b 2d 64 38  37 35 3
0090  33 31 32 36 35 63 34 36  39 36 6
00a0  2d 64 38 37 35 34 33 2d  3b 72 7
00b0  4d 61 78 2d 46 6f 72 77  61 72 6
00c0  0d 0a 43 6f 6e 74 61 63  74 3a 2
00d0  32 30 30 30 31 40 31 36  33 2e 3
00e0  31 35 32 3a 36 33 32 31  32 3e 0
00f0  22 32 30 30 30 33 22 3c  73 69 7
0100  33 40 31 36 33 2e 32 32  2e 32 3
0110  0d 0a 46 72 6f 6d 3a 20  22 53 6
0120  20 75 73 69 6e 67 20 58  2d 4c 6
0130  30 22 3c 73 69 70 3a 32  30 30 30 31 40 31 36 33   0"<sip:2 0001@163

File: "C:\DOCUME~1\ADMINI~1\L...   P: 15 D: 15 M: 0 Drops: 0
```

```
Time     163.22.20.152                          163.22.20.154   Comment
0.000    INVITE SDP ( BV32 BV32-FEC g711U iLB  (63212) -> (5060)   SIP From: sip:20001@163.22.20.154 To:sip:200
0.003    100 Trying         (63212) <- (5060)   SIP Status
1.310    180 Ringing        (63212) <- (5060)   SIP Status
1.509    180 Ringing        (63212) <- (5060)   SIP Status
3.753    CANCEL             (63212) -> (5060)   SIP Request
3.758    200 OK             (63212) <- (5060)   SIP Status
4.588    487 Terminated     (63212) <- (5060)   SIP Status
4.589    ACK                (63212) -> (5060)   SIP Request
9.219    INVITE SDP ( BV32 BV32-FEC g711U iLB  (63212) -> (5060)   SIP From: sip:20001@163.22.20.154 To:sip:200
9.222    100 Trying         (63212) <- (5060)   SIP Status
12.087   180 Ringing        (63212) <- (5060)   SIP Status
13.942   200 OK SDP ( g711U) (63212) <- (5060)  SIP Status
14.408   ACK                (63212) -> (5060)   SIP Request
17.680   BYE                (63212) <- (5060)   SIP Request
17.859   200 OK             (63212) -> (5060)   SIP Status
```

Figure 1. Graphical user interface of Ethereal to show the captured packets and SIP message flow

Users may use web browsers to show a specific SIP message in plaintext. In Distributed SIP Analyzer, BSD Packet Filter (BPF) [5] is utilized to capture SIP messages and store them in MySQL database. Users may use web browsers to query the database to obtain the result from the server after running a PHP script. To sum up, in both implementations, the server captures the SIP messages from its network interface and store them in a pre-defined format for client browsers to query. Since users are only allowed to query via a web browser, it is not necessary to give them the super-user privilege on the server. A normal user privilege is enough. When it is necessary, the system administrator can also apply the access control mechanism that is commonly adopted in web administration, to further limit what SIP messages can be seen for different levels of user accounts. This provides better protection of the sensitive information of telephony call records.

However, due to the traditional limitation of web applications, these two tools are not capable of showing the captured SIP messages in real time. To get the most up-to-date information on the server, a webpage must be refreshed, and the user must wait for the response while the server is handling the request. In comparison with Ethereal, which provides better interaction with users, aforementioned web tools are less inconvenient. In this paper we present the design and implementation of a web-base SIP analyzer which is capable of always showing the up-to-date results of packet capturing. By integrating the Asynchronous JavaScript and XML (Ajax) approach into our system, we are able to update the webpage without a browser refresh. This makes it look and act more like desktop applications, since users do not spend time waiting for entire pages to reload, which is essentially a common drawback in traditional web applications.

The rest of this paper is organized as follows. Section 2 describes the design of our system architecture, and the functionality of each components is illustrated. Section 3 provides the implementation notes for integrating these components. Section 4 presents some security and privacy issues, and Section 5 concludes the paper.

2 System architecture

Figure 2 depicts the overview of our web-base SIP analyzer. The system consists of two portions: a group of components on the SIP server to capture the SIP messages and store them in a database for future query, and the other group of components on web browser client to dynamically update the results displayed to users. We shall illustrate the function of each component and supporting technologies in the following subsections.

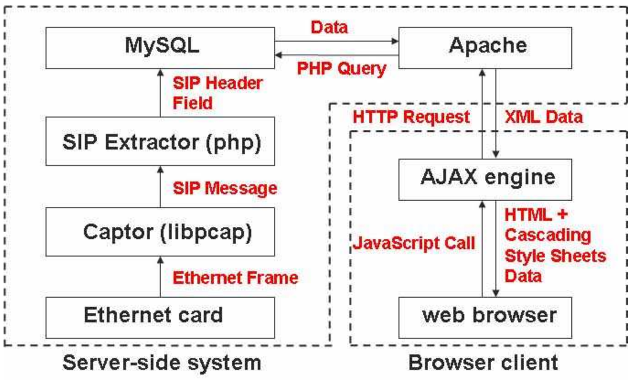


Figure 2. Architecture of our Web-base SIP analyzer

2.1 Server-side system

For packets sent and received on the Ethernet interface of the SIP server, we run a C program *Captor* invoking the *libpcap* library [6] to capture these packets. This library provides a common interface on various operating systems (including FreeBSD, Linux, MS-Windows) to perform packets filtering. Programmers can specify the conditions in a high level description (e.g. `udp port 5060`) and call the function *pcap_setfilter()* to ask the kernel of the operating system to perform the packet filtering. A function *pcap_loop* is invoked to iterate through for all packets captured by the kernel. We then define a callback function *got_packet()* to handle these captured SIP messages. We strip off the IP and UDP headers and output the SIP header fields with a function *print_payload()*. This output is piped to another process running a PHP script *SIP_Extractor* to parse the SIP header fields and store them into the MySQL database. Because PHP supports *associative arrays*, and provides rich functions in manipulating strings, it is an ideal tool to handle this task. Note that in comparison with [1, 7], the architectural design of the server portion is very similar, except different tools were chosen in each implementation.

2.2 Client system

The real-time update on the browser client is less straightforward. With typical web applications, users must spend time waiting for entire pages to reload, even for small changed. This makes the applications less interactive, and usually frustrates users with the long and frequent waiting. Asynchronous JavaScript and XML (Ajax) is a web development technique for creating interactive web applications [3]. It can add or retrieve new data for a webpage and update the page immediately without reloading. As shown in Figure 3(a), in a traditional web application, the user action (generally a click on the SUBMIT button) triggers an HTTP request to a web server. The server will processes the request and returns an HTML page to the client. While the browser is waiting for the response from the server to update the page, the application is locked up and no additional request is handled during this waiting time. On the contrary, Ajax has a different approach to handle requests. As depicted in Figure 3(b), by creating a JavaScript-based engine that runs on the browser, the engine takes user inputs and handles many interactions on the client side. If the engine needs more data, it utilizes the browser object *XMLHttpRequest* to request data from the server in the background, while letting users continue to interact with the applications. This provides better interaction which allows the user to feel as though they are manipulating desktop-resident software. Therefore, this approach recently becomes extremely popular. Ajax is supported by major browsers, including Internet Explorer since 5.0, Mozilla since 1.0, Safari since 1.2, and Opera since 8.0. Many gorgeous websites, such as Google Maps (http://maps.google.com/) and Google Suggest (http://www.google.com/webhp?complete=1) demonstrates how Ajax can be helpful in improving the interactivity of web applications.

In our client-side JavaScript code, the Ajax engine will get data from the web server. While users are viewing the SIP messages displayed on their screen, the Ajax engine keeps communicating with the web server asynchronously. Whenever it notices that additional SIP messages are captured by the server, it requests them asynchronously from the server. After it receives all the incoming data, it updates the user screen. Therefore, the user screen is updated in a short time. Compared with Ajax, classic web applications keep users waiting whenever they send a request to the server. For these application to show the most up-to-date results, only ”periodic update” (e.g. every one minute) can be implemented. If the webpage was reloaded every second, users will notice the frequent application lock up, which is generally intolerable.

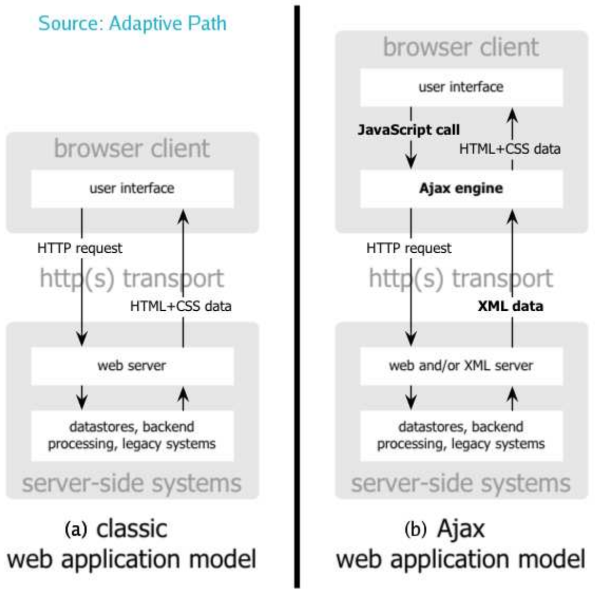
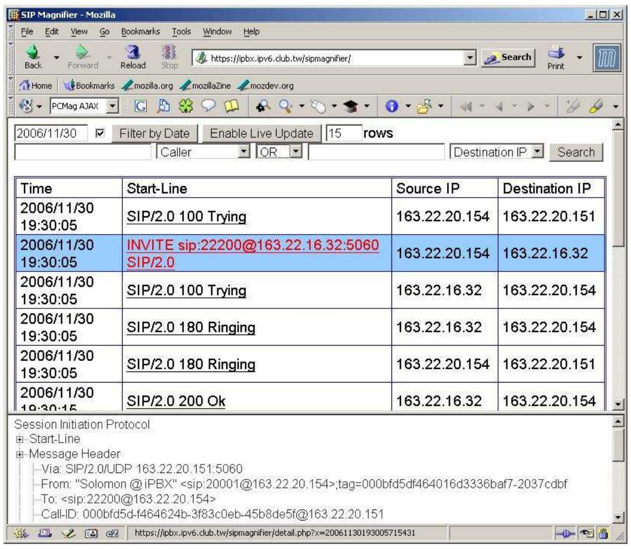


Figure 3. The traditional model for web applications compared to the Ajax model

After the Ajax engine receives the SIP messages from the server, it organizes them in an HTML table and show some major fields of interest, such as the timestamp of the packet, the SIP method name or response code, and the source/destination IP addresses of the SIP message. The SIP method name can be the standard ones defined in [9], such as INVITE, REGISTER, CANCEL, OPTIONS, BYE, ACK. It can also be any extension like SUBSCRIBE, NOTIFY, INFO. Our parser can handle any SIP message with correct syntax. It is unnecessary to modify our analyzer in order to handle any new SIP extension. When users click a SIP message, the full contents of the SIP header will be displayed in another frame, and the user can choose to expand or collapse the display of SIP start line, message header, or message body (generally to be the Session Description Protocol [4] which specifies the connection address and media format), as shown in Figure 4.


SIP Magnifier - Mozilla
File Edit View Go Bookmarks Tools Window Help
Back Forward Reload Stop
https://ipbx.ipv6.club.tw/sipmagnifier/
Search
Print
Home Bookmarks mozilla.org mozillaZine mozdev.org
PCMag AJAX

2006/11/30 ☑ Filter by Date | Enable Live Update | 15 rows

Caller | OR | Destination IP | Search

Time	Start-Line	Source IP	Destination IP
2006/11/30 19:30:05	SIP/2.0 100 Trying	163.22.20.154	163.22.20.151
2006/11/30 19:30:05	INVITE sip:22200@163.22.16.32:5060 SIP/2.0	163.22.20.154	163.22.16.32
2006/11/30 19:30:05	SIP/2.0 100 Trying	163.22.16.32	163.22.20.154
2006/11/30 19:30:05	SIP/2.0 180 Ringing	163.22.16.32	163.22.20.154
2006/11/30 19:30:05	SIP/2.0 180 Ringing	163.22.20.154	163.22.20.151
2006/11/30 19:30:15	SIP/2.0 200 Ok	163.22.16.32	163.22.20.154

```
Session Initiation Protocol
⊞ Start-Line
⊞ Message Header
   Via: SIP/2.0/UDP 163.22.20.151:5060
   From: "Solomon @ iPBX" <sip:20001@163.22.20.154>;tag=000bfd5df464016d3336baf7-2037cdbf
   To: <sip:22200@163.22.20.154>
   Call-ID: 000bfd5d-f464624b-3f83c0eb-45b8de5f@163.22.20.151
```

https://ipbx.ipv6.club.tw/sipmagnifier/detail.php?x=20061130193005715431

Figure 4. Captured SIP messages and the detailed contents of a message

3 Implementation notes

According to [9], the default SIP port is 5060 running on UDP, so the default capture filter in our system is configured as ”`udp port 5060`”. It can be configured to capture packets on other UDP ports or on other transport protocols like TCP or SCTP, depending on what is the port and transport protocol used by the SIP server on this host. Although *libpcap* supports a *promiscuous* mode to have the operating system kernel capturing all packets on the Ethernet link, in our application we only care about the SIP messages sent and received on the SIP server. Therefore, we disable the promiscuous mode when we invoke the *libpcap* library.

When the client queries the database to look for SIP messages of interest, we allow users to get all the SIP messages in a single day, originated from or destined to a specific IP address, or only shows a specific type of SIP request (REGISTER, INVITE, BYE, and so forth). The user may also click the ”Enable Live Update” button to ask the browser to asynchronously get the most up-to-date data from the server, and show them on the user screen whenever there comes new information. To display those messages more elegantly and more smoothly, we place a limit on the display records when the browser is running in the ”Live Update” mode. In our example we specify that only the most recent 15 SIP messages are displayed, but this can be adjusted by the web form before enabling the ”Live Update” mode.

This tool is published as open source on SourceForce (https://sourceforge.net/projects/ncnu-voip). We wish this application could add an example to the exciting services which have been demonstrated by the Ajax approach.

4 Security and privacy issues

Conventionally, if you want to enable a user to view the SIP signals on a server, you need to give him/her the superuser privilege to run Ethereal, because Ethereal needs the privilege to access the network interface to capture packets. With our tool, packets are captured and deposited into the database, so further access can be controlled via the database system or web server. Any user who wants to view the SIP messages will only get normal user privilege, but he/she is able to run our SIP analyzer to observe the real-time SIP signaling, which is very useful in trouble-shooting. This is an improvement on security by limiting the user privilege in accessing data, while the convenience is not sacrificed.

However, for some sensitive sites, it is also possible that you want to limit the SIP messages displayed on the screen. It is reasonable that an ITSP does not want the engineer from a WiFi phone vendor to be able to see all the calling records between its customers during an interoperability test. In this case, our approach can enforce the extra limit by applying access control mechanisms of web server or database management system, which is certainly a better choice in comparison with a tool like Ethereal because Ethereal by nature allows the user to view any packet on the server.

5 Conclusions and future work

As more and more SIP appliances emerge, web-base SIP analyzer is a convenient tool for developers, especially for trouble-shooting embedded systems such as WiFi phones. By adopting Ajax approach, our system can continuously update the user screen without keeping user waiting, which makes it act more like desktop applications.

Although this system proves to be convenient and useful in a campus SIP VoIP trial, it is not clear whether the Ajax engine will consume extra resource that will cause any performance issue. We expect to perform further pressure tests to study the maximum number of SIP messages that can be handled by our system, and compare it with other analysis tools.

References

[1] Devel-IT Team. WIST - web interface for SIP trace. *http://sourceforge.net/projects/wist/*.

[2] Ethereal, Inc. Ethereal: A network protocol analyzer. *http://www.ethereal.com/*.

[3] J. J. Garrett. Ajax: A new approach to web applications. *http://www.adaptivepath.com/publications/essays/archives/000385.php*.

[4] M. Handley, V. Jacobson, and C. Perkins. SDP: Session description protocol. *IEEE RFC 4566*, July 2006.

[5] S. McCanne and V. Jacobson. The BSD packet filter: A new architecture for user-level packet capture. In *Proceedings of the 1993 Winter USENIX Technical Conference*, pages 259–269, San Diego, CA, January 1993.

[6] S. McCanne, C. Leres, and V. Jacobson. Libpcap. *http://www.tcpdump.org/*.

[7] J.-Y. Pan. Distributed SIP analyzer. *http://sourceforge.net/projects/sipanalzyer/*.

[8] J. Pulver. SIP products list. *http://www.pulver.com/products/sip/*.

[9] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session initiation protocol. *IEEE RFC 3261*, June 2002.