|  |
| --- |
|  |
| SIP Sorcery |
| Experimental SIP Software |

|  |
| --- |
| Aaron Clauson  4/21/2009 |

Contents

[1.0 Overview 3](#_Toc231429943)

[1.1 Deployment Scenarios 3](#_Toc231429944)

[2.0 SIP Server Agents 4](#_Toc231429945)

[2.1 Application Server 4](#_Toc231429946)

[2.2 Monitor Server 4](#_Toc231429947)

[2.3 Stateless Proxy 4](#_Toc231429948)

[2.3.1 Loop Detection 4](#_Toc231429949)

[2.3.2 Multiple SIP Sockets 5](#_Toc231429950)

[2.3.3 Example Usage Scenarios 5](#_Toc231429951)

[2.4 Registrar 7](#_Toc231429952)

[2.5 Registration Agent 8](#_Toc231429953)

[3.0 How to Use 8](#_Toc231429954)

[4.0 Troubleshooting 9](#_Toc231429955)

[4.1 Process does not have access rights for web service 9](#_Toc231429956)

# 1.0 Overview

The SIP Sorcery project revolves around a SIP protocol stack and a set of experimental SIP server applications built on top of it. The SIP Sorcery project has evolved out of an earlier project called MySIPSwitch which has been running as a publicly accessible SIP service from Nov 2006. SIP Sorcery is predominantly a name change for the MySIPSwitch project but has been combined with a major software upgrade which has included enhancements to the SIP Stack, the SIP Server Agents and has also added a new rich client interface based on Silverlight to replace the previous HTML and Javascript based interface.

The SIP Sorcery software is all based on C# and version 3.5 SP1 of the .Net framework. The various assemblies that make up the project fall into one of four main categories:

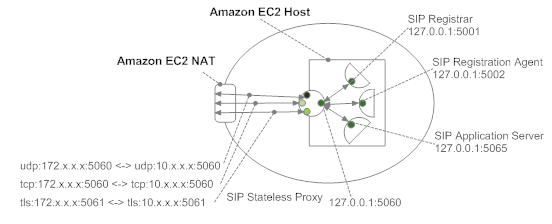
* Core -This category indicates an assembly is part of the SIP Sorcery “plumbing”. It includes the assembly which implements the SIP protocol stack, **SIPSorcery.SIP.Core**, as well as five other assemblies that include implementations for persistence, network services such as DNS, application services for managing resources such as SIP Accounts, SIP Providers and Customer records,
* Servers – This category indicates an assembly is a SIP Server Agent and represents an application that can be run to provide a specific SIP service such as a Stateless SIP Proxy or SIP Registrar,
* Silverlight – This category indicates the assembly is part of the new Silverlight user interface and will be run within the Silverlight plugin inside a web browser,
* SoftPhone – This category indicates the assembly is part of a new SoftPhone which has been implemented predominantly as a test tool for the Core and Server parts of the project. The SoftPhone is extremely rudimentary and does not implement any audio processing.

## 1.1 Deployment Scenarios

The SIP Sorcery Server Agents can be deployed in a variety of different IP network environments. Examples of some different ones are:

* Typical NAT’ted and PAT’ted networks such as ones from residential ISPs,
* The NAT’ted but non-PAT’ted network of the Amazon EC2 cloud,
* Public IP address.

The deployment approach used depends mainly on the configuration required by the SIP Transport for the SIP Proxy Server and possibly the SIP Application Server.



Deployment Diagram – Amazon Cloud

# 2.0 SIP Server Agents

## 2.1 Application Server

The Application Server is a SIP B2BUA which can process SIP call traffic based on user customisable dial plans. The Application Server is designed to only deal with call related SIP messages (ACK, BYE, CANCEL and INVITE) and it will reject any other requests.

For new INVITE requests the Application Server will utilise persisted SIP Assets: Customer Records, SIP Accounts, SIP Dial Plans, SIP Providers etc; to provide sophisticated call management. At the heart of this call management is a dial plan which can be written as either an Asterisk compatible line-by-line dial plan or a much more powerful and flexible Ruby script dial plan. Each of the dial plans can make use of inbuilt applications such as Dial to forward a call, Respond to reject a call, GTalk to send an instant message to a Google Talk user and many more.

## 2.2 Monitor Server

The Monitor Server is a monitoring agent that captures log messages from other server agents and that also provides a telnet interface for end users to connect and view the log messages. The purpose of the Monitor Server is to provide a simple way to get visibility into the actions being taken by the other SIP Sorcery Server Agents. The Monitor Server requires connecting users to specify a filter which is used to restrict the events that will be sent to the user’s telnet connection.

## 2.3 Stateless Proxy

The SIP Proxy Server is a stateless SIP Proxy that utilises either a Ruby or Python script to control SIP message flows. The Proxy Server is designed to sit in front of all the other SIP Sorcery SIP Servers and forward incoming SIP requests to the appropriate Server Agent: INVITE’s to the Application Server, REGISTER’s to the SIP Registrar etc.

The SIP Proxy also handles any packet mangling that may be required if it is operating in a NAT environment. If multiple SIP sockets are used the SIP Proxy can be configured to track the flow of requests depending on requirements.

### 2.3.1 Loop Detection

In order to be able to detect looped SIP requests the Proxy needs a mechanism to determine whether a request it has received has both previously been processed by the Proxy AND none of the fields which influence the request’s routing have changed in which case the request has spiralled rather than looped.

The approach used to achieve this is that the Proxy host code will generate the branch parameter and make it available for the runtime script. It’s then up to the script to pass the branch parameter when forwarding the request. There are certain cases where the branch parameter set by the Proxy host will not be used. For example when forwarding requests from the SIP Registration Agent, which can only contain a single **Via** header, the Proxy will copy the branch parameter from the Registration Agent’s request rather than generating a new one.

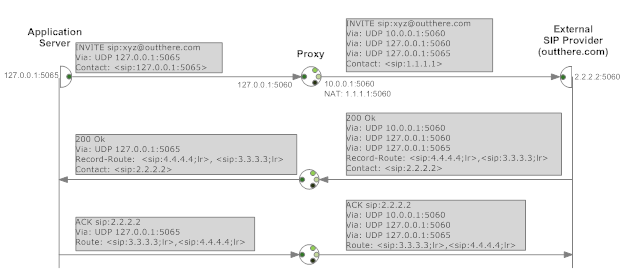
### 2.3.2 Multiple SIP Sockets

When the SIP Proxy is configured with multiple SIP sockets its operation becomes more complex as the same SIP message can be received and forwarded on a different socket. When the occurs a standard **Via** header will not contain enough information to allow the SIP response to travel back on the reverse path through the Proxy.

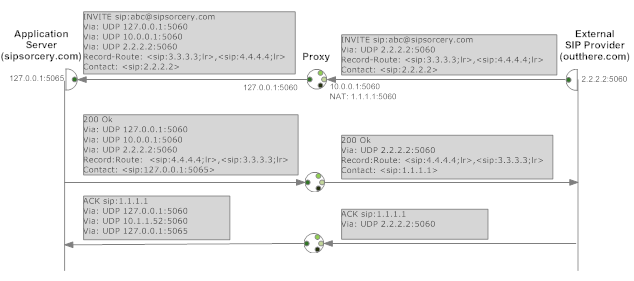
### 2.3.3 Example Usage Scenarios

#### 2.3.3.1 Outbound Proxy

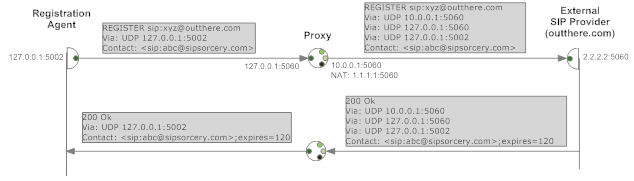
An outbound proxy scenario could be deployed to allow a private SIP Server Agent operating on a private or loopback address to communicate with external SIP Agents. By sending all its requests via the outbound proxy the private SIP Server could operate in a more controlled environment and rely on the Proxy for a number of services such as security, SIP message mangling, SIP message validation etc.



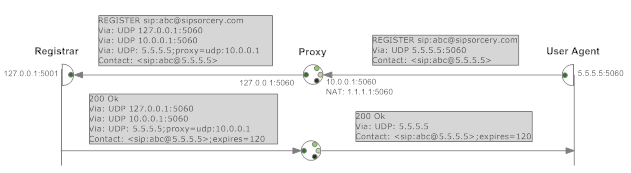
SIP Ladder Diagram – Outgoing Call with Outbound Proxy



SIP Ladder Diagram – Incoming Call with Outbound Proxy



SIP Ladder Diagram – Registration Agent with Outbound Proxy



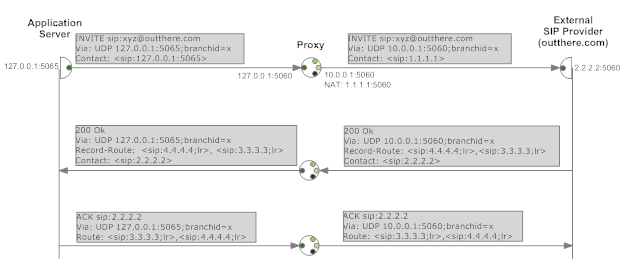
SIP Ladder Diagram – Registrar with Outbound Proxy

#### 2.3.3.2Transparent Outbound Proxy

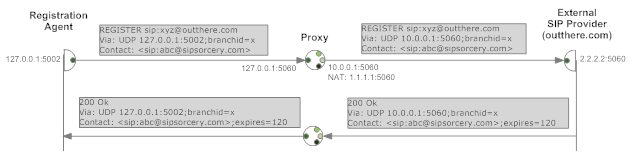
A Transparent Outbound Proxy mode of operation is identical to an Outbound Proxy mode except that the proxy does not insert a **Via** header when forwarding requests. This mode of operation is not compliant with the SIP standard; chapter 16.6 section 8 mandates SIP Proxies must insert a **Via** header when forwarding requests.

The requirement for this mode of operation stems from real World experience where a number of SIP Providers have been found to reject requests that have more than one **Via** header. In most cases the Providers are deliberately rejecting the requests in order to prevent end users routing traffic through SIP Proxies and possibly circumventing IP address checks; the Betamax family of services have been known to operate in this fashion. In at least one other case a SIP Provider (Callcentric) seems to inadvertently reject REGISTER requests with multiple **Via** headers. If the REGISTER request is delivered with multiple **Via** headers a “500 Network Error” response is returned but the exact same request with only a single Via header is accepted.

Since the primary objective of the SIP Sorcery service is to facilitate as wide a possible inter-operability with SIP Providers it is worthwhile accommodating the Providers that only accept single **Via** header requests. It is only possible for the SIP Sorcery Proxy to operate in a Transparent Outbound Proxy mode when operating with a specific transport layer configuration namely that the Proxy can unambiguously determine the socket on which to forward each response without relying on a dedicated **Via** header.



SIP Ladder Diagram – Outgoing Call with Transparent Outbound Proxy



SIP Ladder Diagram – Registration Agent with Transparent Outbound Proxy

Only requests that originate from SIP Sorcery server agents need to utilise transparent proxy mode. The ladder diagrams for incoming calls and the Registrar are the same as for the Outbound Proxy case.

Requests from external SIP Providers or clients can still use multiple **Via** headers to ensure the correct path is followed back through the Proxy. It would be more difficult to operate in transparent mode with external requests since the Proxy will be receiving requests on multiple sockets whereas it will only ever communicate with the other SIP Sorcery server agents on a single socket.

## 2.4 Registrar

The SIP Registrar is a standard SIP Registrar that processes user agent binding requests.

## 2.5 Registration Agent

The SIP Registration Agent can be thought of as the opposite of a SIP Registrar, it sends binding requests to external SIP Registrars.

# 3.0 How to Use

The SIP Server Agents provide a number of pre-canned applications that can be used as is or tweaked as needed. For different SIP applications where there is no pre-existing agent the SIP protocol stack can be quickly and easily incorporated.

In order to use the SIP stack all that’s needed is a reference to **SIPSorcery.SIP.Core**. After that to make use of the stack there are a small number of SIP concepts that need to be understood so that it can be configured to operate in the correct way for an application.

* The SIP standard specifies a number of different network and transport layer protocols that SIP can operate on. The SIP Sorcery stack supports UDP, TCP and TLS. When configuring the SIPSorcery SIP stack the first step is to inform it on which and what type of communications endpoints it will be using,
* SIP uses transactions to link together requests and responses. Most SIP agents do need to operate with transactions but some such as Stateless SIP Proxy’s do not. The SIPSorcery SIP stack can operate in either mode but if transactions are not required it is far better to deactivate the transaction engine to reduce the computing resources used by the SIP stack.

As an example application a code sample is provided that sends a periodic OPTIONS request to a SIP agent to check whether it is alive.

SIPUDPChannel demoChannel = new SIPUDPChannel(new IPEndPoint(IPAddress.Loopback, 5090));

SIPTransport sipTransport = new SIPTransport(SIPDNSManager.Resolve,

new SIPTransactionEngine(), demoChannel, true, false);

int cseq = 1;

while (true) {

SIPHeader header = new SIPHeader("<sip:127.0.0.1:5090>", "<sip:127.0.0.1>", cseq,

CallProperties.CreateNewCallId());

SIPRequest optionsRequest = new SIPRequest(SIPMethodsEnum.OPTIONS,

SIPURI.ParseSIPURIRelaxed("127.0.0.1")) { Header = header };

header.CSeqMethod = SIPMethodsEnum.OPTIONS;

SIPViaHeader viaHeader = new SIPViaHeader(demoChannel.SIPChannelEndPoint,

CallProperties.CreateBranchId());

optionsRequest.Header.Vias = new SIPViaSet();

optionsRequest.Header.Vias.PushViaHeader(viaHeader);

SIPNonInviteTransaction optionsTransaction = sipTransport.CreateNonInviteTransaction(optionsRequest, sipTransport.GetRequestEndPoint(optionsRequest, null, true), demoChannel.SIPChannelEndPoint, null);

optionsTransaction.NonInviteTransactionFinalResponseReceived += (l, r, t, resp) => {Console.WriteLine(" response " + resp.StatusCode + " received for cseq=" + resp.Header.CSeq + ".");};

optionsTransaction.NonInviteTransactionRequestRetransmit += (t, r, n) => { Console.WriteLine(" request cseq=" + r.Header.CSeq + " retransmit number=" + n + "."); };

optionsTransaction.NonInviteTransactionTimedOut += (t) => { Console.WriteLine(" response timed out for cseq=" + t.TransactionRequest.Header.CSeq + "."); };

Console.WriteLine(" sending OPTIONS request, cseq=" + optionsRequest.Header.CSeq + ".");

optionsTransaction.SendReliableRequest();

Thread.Sleep(40000);

cseq++;

}

Code Sample – Example of using SIPSorcery.Core

# 4.0 Troubleshooting

## 4.1 Process does not have access rights for web service

Execute the command below from a command prompt with administrative permissions (use runas to start the DOS box).

For Windows Vista:

**netsh http add urlacl url=http://+:8080/ user=DOMAIN\user**

For Windows XP and 2003:

httpcfg set urlacl /u <http://+:8080/callmanager> /a D:(A;;GX;;;LS)

Also search for tool HttpCfgACL to make life easier with the security descriptor syntax.