San José State University

EE- 284

CONVERGENT VOICE AND DATA NETWORKS

SPRING 2018

COURSE PROJECT- 1

A HANDS-ON SIP-BASED VOIP EXPERIMENTS ON: CALL ESTABLISHMENT, BUSY LINES, CALL ON HOLD, AND CONFERENCE CALLING

GROUP- 4

PARVATHI CHANDRASEKHAR (012474995)

VARSHA AJAY KOHIRKAR (012031643)

GAOJUN SU (012465518)

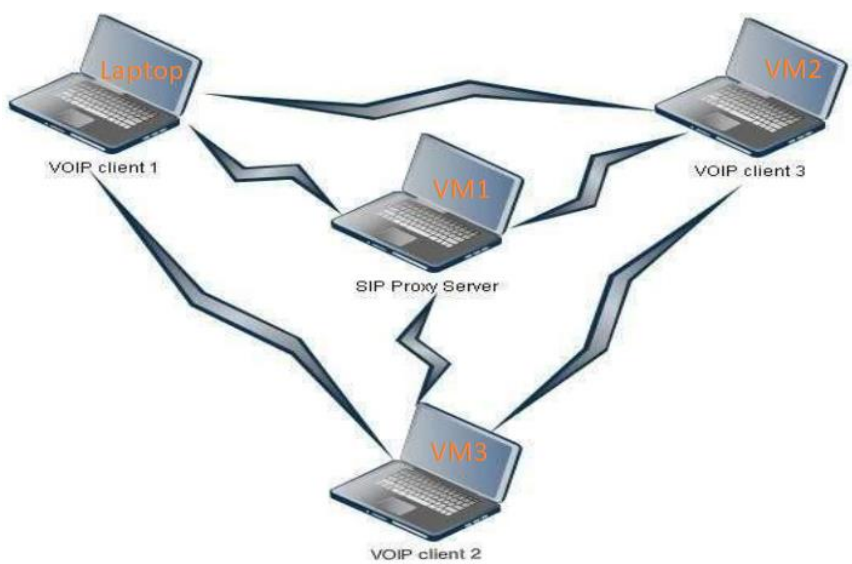
PART-1

Abstract

The aim of the project is to understand the implementation of SIP [Session Initiation Protocol] to enable media transfer or voice communication. The project comprises of one server and three clients to implement the four phases. The server software used is Asterisk Software and the clients use X-Lite soft phones to implement voice calls. Client registers with the server and calls are then established. Four different call scenarios are observed and implemented. The call scenario packets are screen captured using Wireshark and call flows are also observed.

HARDWARE SETUP AND TOOL DOWNLOADS:

1. Call Scenarios



Phase 1 and 2 of this project only need 2 clients. Phase 1 includes a simple call. Phase 2 comprises of a call invite from one request but busy tone sent from the other client. Phase 3 and 4 require a third client. Phase 3 includes a call between two clients, with third client calling and existing call put on hold and third client's call picked up. Phase 4 is a conference call where a call is already established between 2 clients and then the third client calls and the calls are merged.

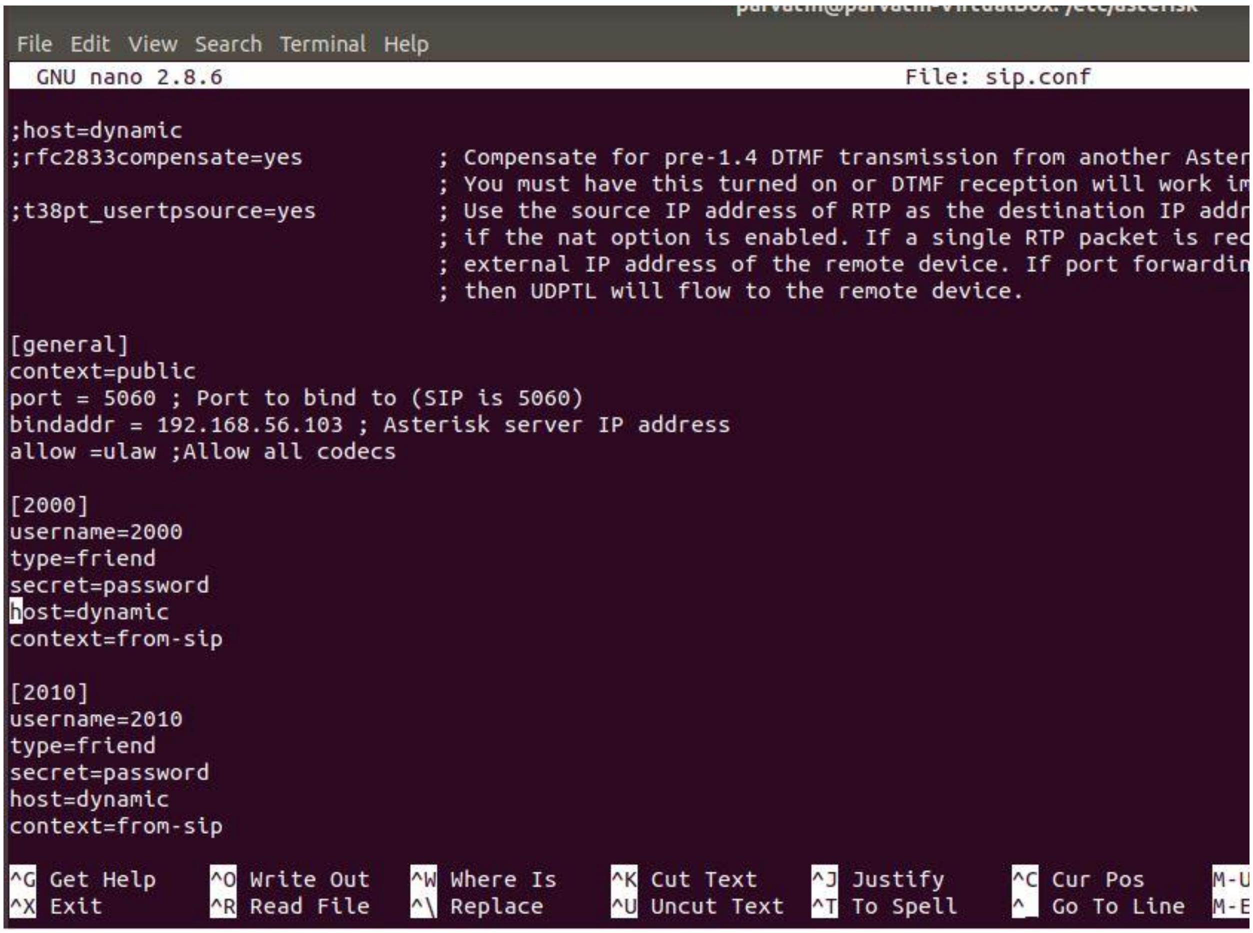
2. Network Setup

This project comprises of one Ubuntu server and three Windows 7 clients. We used VirtualBox to implement the virtual machines, and set the network adapters of each as a Host only network to connect them together in one LAN. The network can also be established through a native Ubuntu OS and three clients in different laptops connected through an ad hoc network.

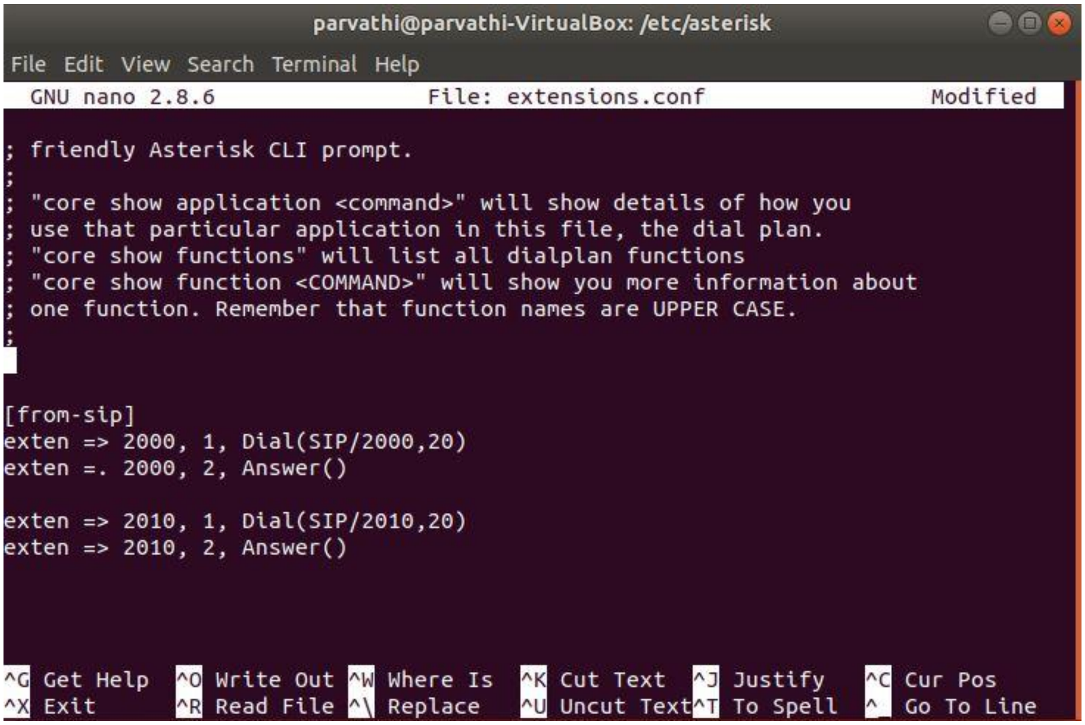
3. Server Setup

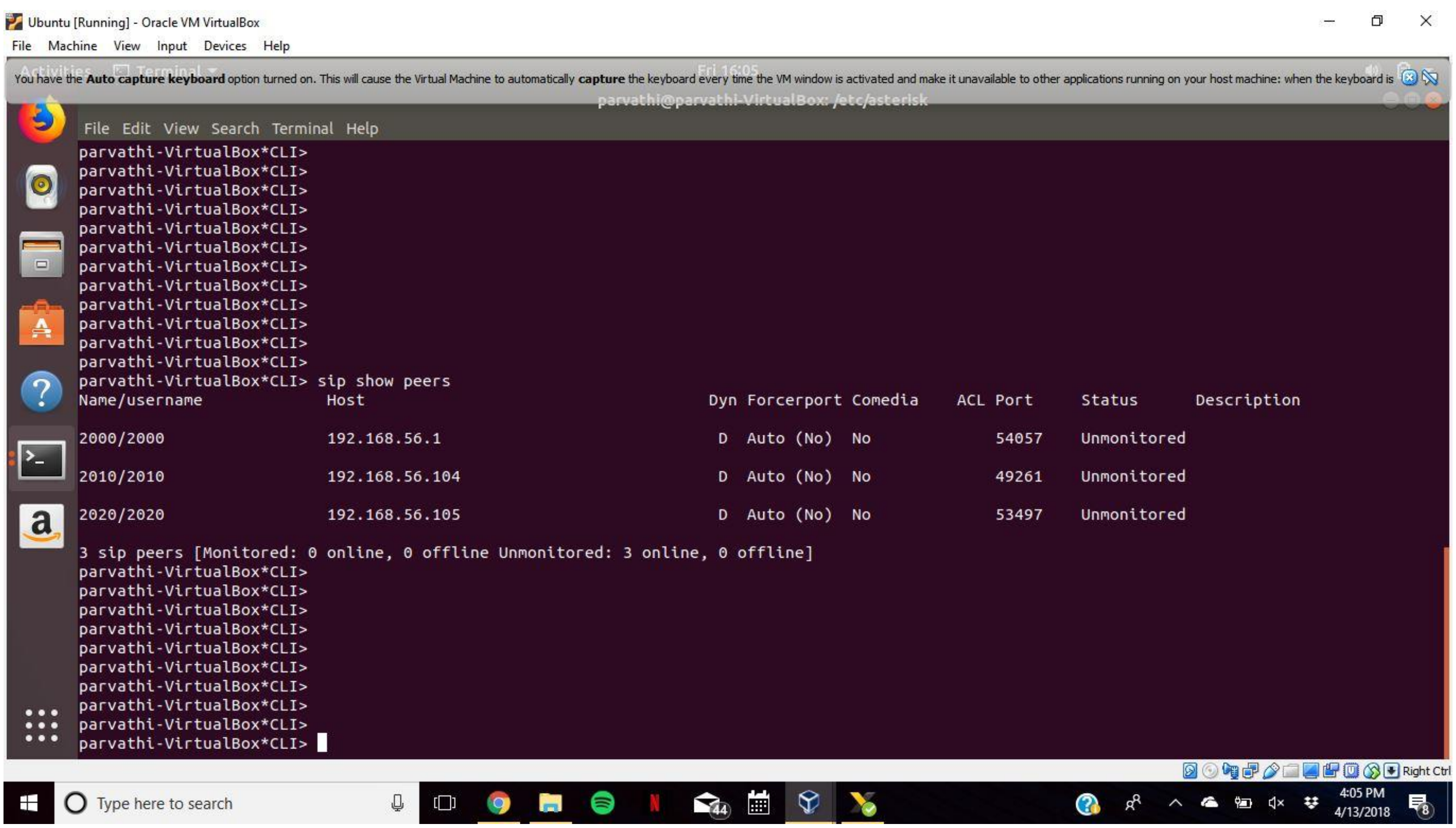
We create the first VM which is the server. We then installed the Asterisk software and all the necessary dependencies for it to work. We modified the sip.conf and extensions.conf file according to the requirements of Phase 1.

We added this to the end of the sip.conf file:



```
File  Edit  View  Search  Terminal  Help
 GNU nano 2.8.6                                               File: sip.conf

;host=dynamic
;rfc2833compensate=yes          ; Compensate for pre-1.4 DTMF transmission from another Aster
                                ; You must have this turned on or DTMF reception will work im
;t38pt_usertpsource=yes         ; Use the source IP address of RTP as the destination IP addr
                                ; if the nat option is enabled. If a single RTP packet is rec
                                ; external IP address of the remote device. If port forwardin
                                ; then UDPTL will flow to the remote device.

[general]
context=public
port = 5060 ; Port to bind to (SIP is 5060)
bindaddr = 192.168.56.103 ; Asterisk server IP address
allow =ulaw ;Allow all codecs

[2000]
username=2000
type=friend
secret=password
host=dynamic
context=from-sip

[2010]
username=2010
type=friend
secret=password
host=dynamic
context=from-sip

^G Get Help     ^O Write Out    ^W Where Is     ^K Cut Text     ^J Justify      ^C Cur Pos      M-U
^X Exit         ^R Read File    ^\ Replace      ^U Uncut Text   ^T To Spell     ^_ Go To Line   M-E
```



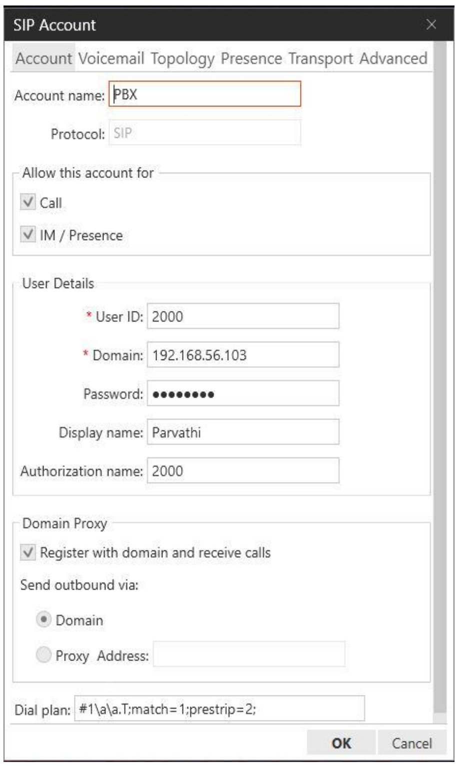


The server and client respective IP addresses are as follows:

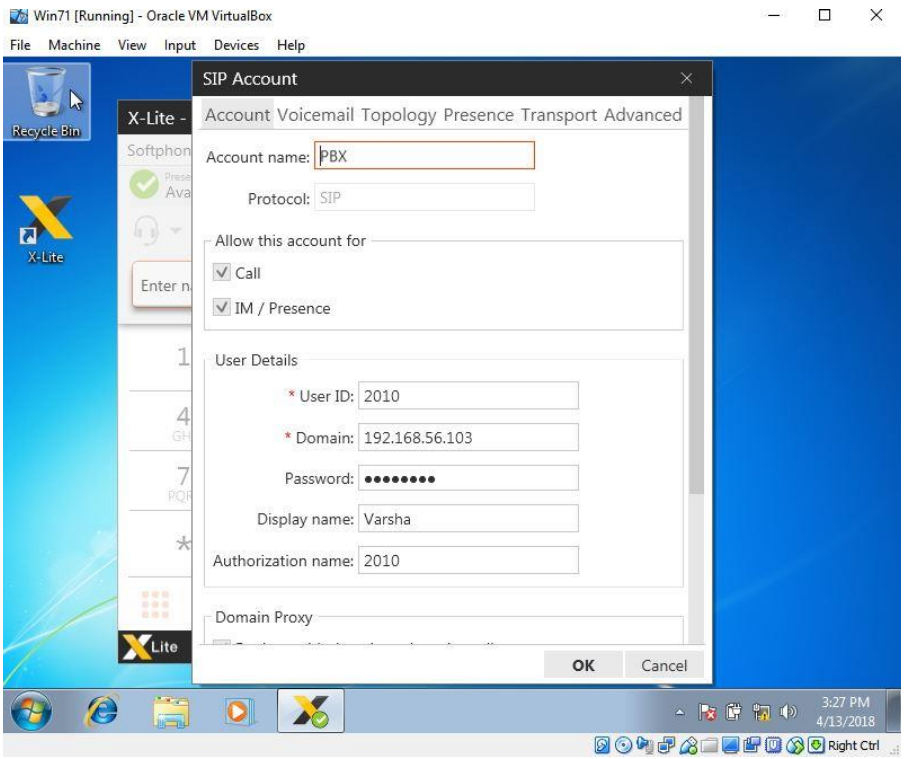
Server or Client	IP Address of S/C
Server	192.168.56.103
Client - I	192.168.56.1
Client - II	192.168.56.104
Client - III	192.168.56.105

The client setup screenshots:

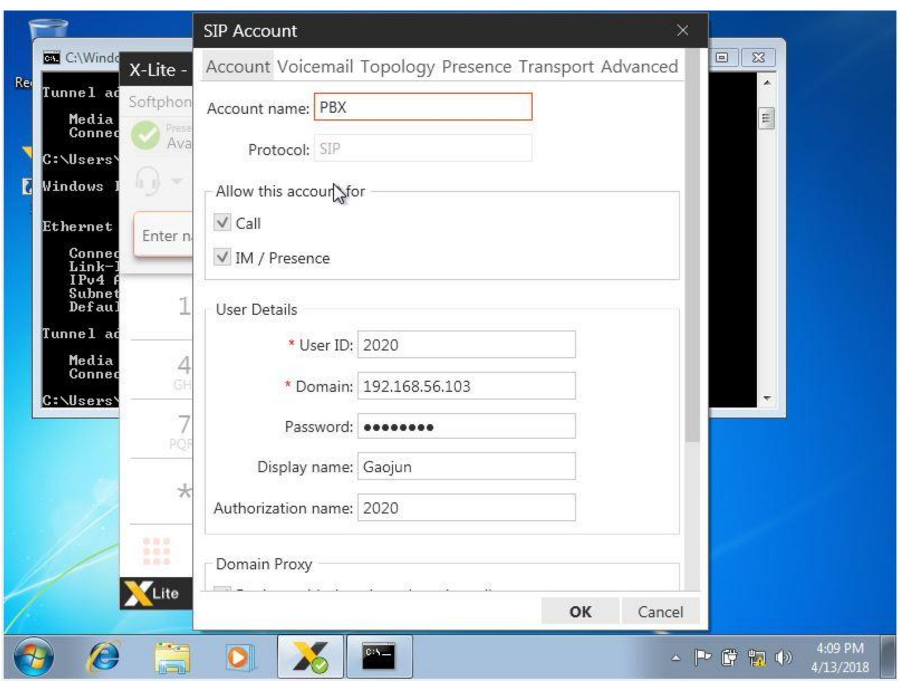
2000 =>



2010 =>



2020 =>



4. Clients Setup

We created 2 more VM's, Client 1 and 2. X-Lite phone were installed next into them and configured according to the instructions given.

5. Connect the clients to the Host-only network.

Connect all the clients with the Proxy Server with the Host-Only Adapter. Assign IP address manually to the Host-Only Adapter in the range "169.254.X.X" with subnet mask "255.255.255.0"

EXPERIMENT:

PHASE 1- ESTABLISH AND ANALYZE A SUCCESSFUL CALL BETWEEN 2 SIP CLIENTS:

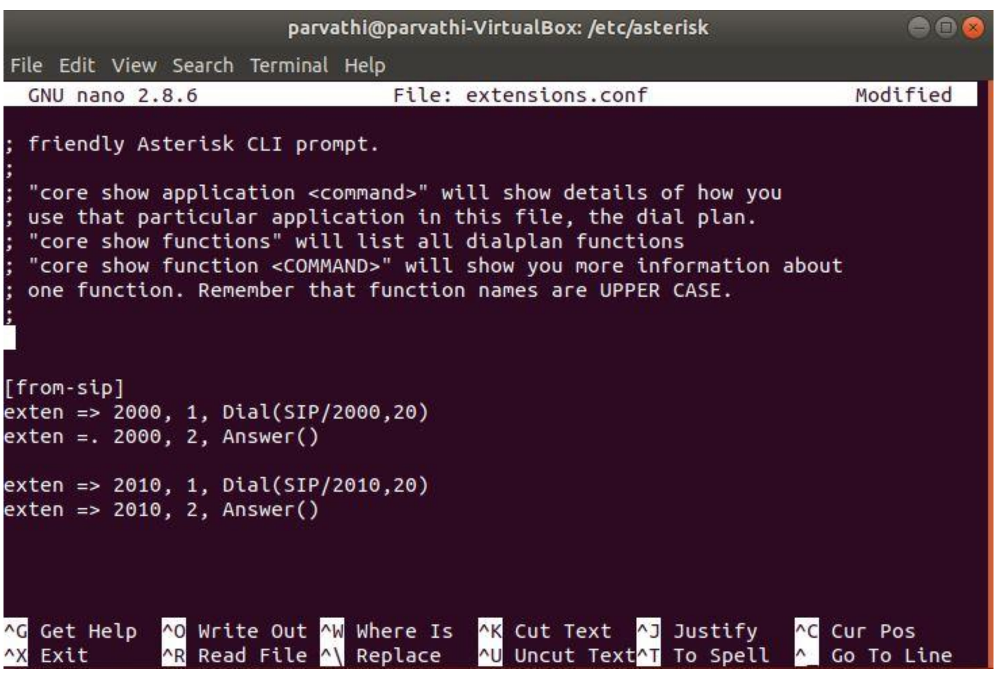
We establish a call between client 1 and 2 after registering both of them to the server.

1. Call establishment.

Call is established between Client 1 and 2. [2000 calls 2010]

The extensions are set as follows:

[from-sip]



```
parvathi@parvathi-VirtualBox: /etc/asterisk
File  Edit  View  Search  Terminal  Help
  GNU nano 2.8.6                File: extensions.conf                Modified

; friendly Asterisk CLI prompt.
;
; "core show application <command>" will show details of how you
; use that particular application in this file, the dial plan.
; "core show functions" will list all dialplan functions
; "core show function <COMMAND>" will show you more information about
; one function. Remember that function names are UPPER CASE.
;


[from-sip]
exten => 2000, 1, Dial(SIP/2000,20)
exten =. 2000, 2, Answer()

exten => 2010, 1, Dial(SIP/2010,20)
exten => 2010, 2, Answer()



^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^_ Go To Line
```

We now type “reload for everything to take effect.

Name on X-Lite Phone	ID
Client I	2000
Client II	2010
Client III	2020

2. Capture of results

Below figure shows the Wireshark capture during the calling between the two clients.

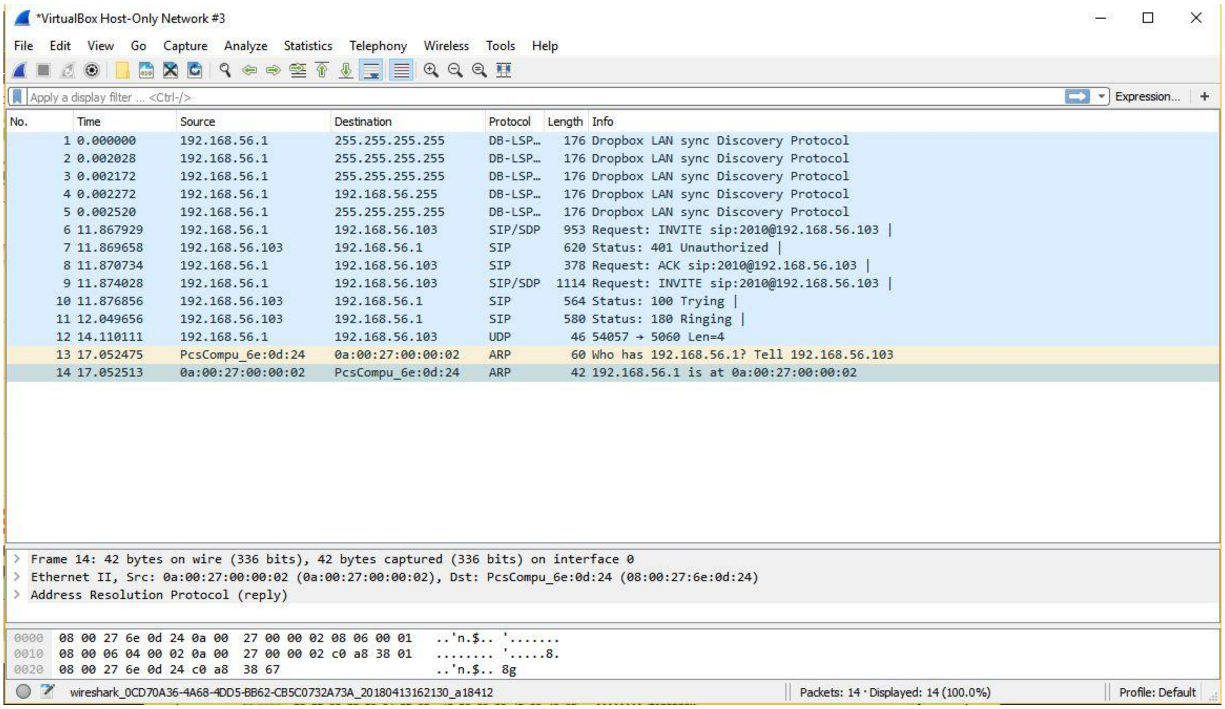


Fig > Calling from one side

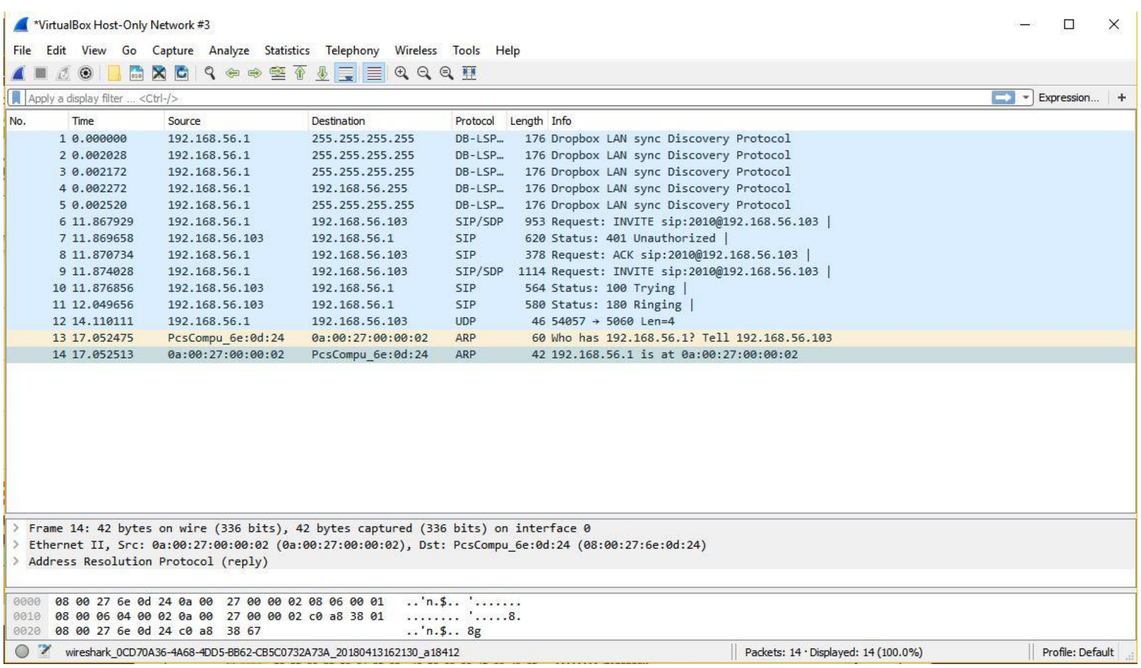


Figure > Call Accepted by the other side

Above Wireshark capture shows the scenario when the call is accepted by the called user.

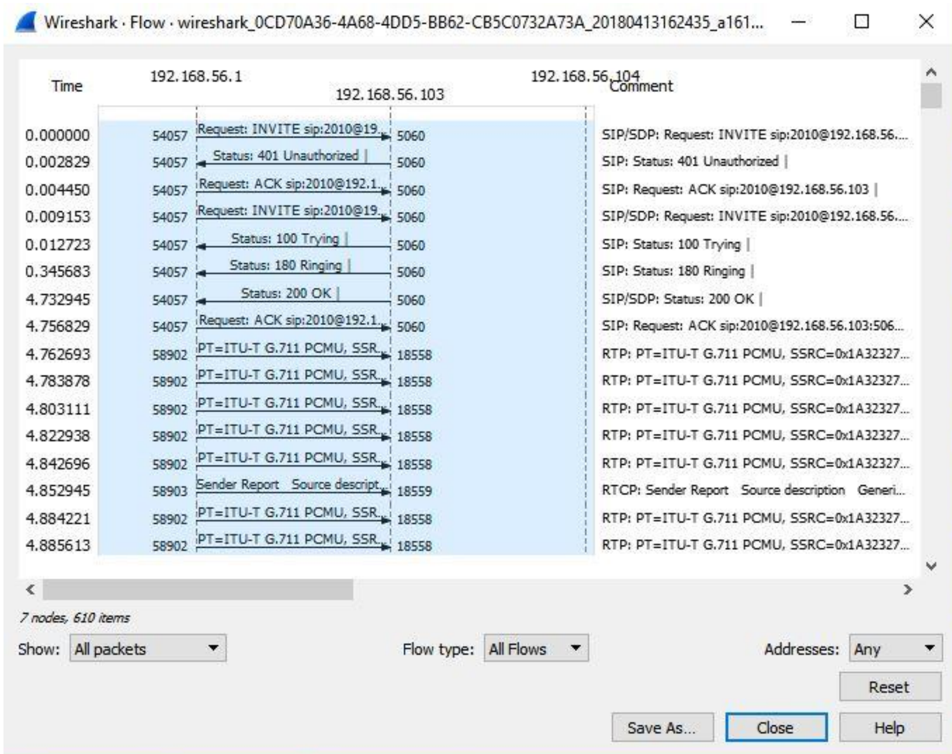


Figure > Call Flow between 2000 and 2010

Above figure shows the capture of message sequences between the 2010 and 2000.

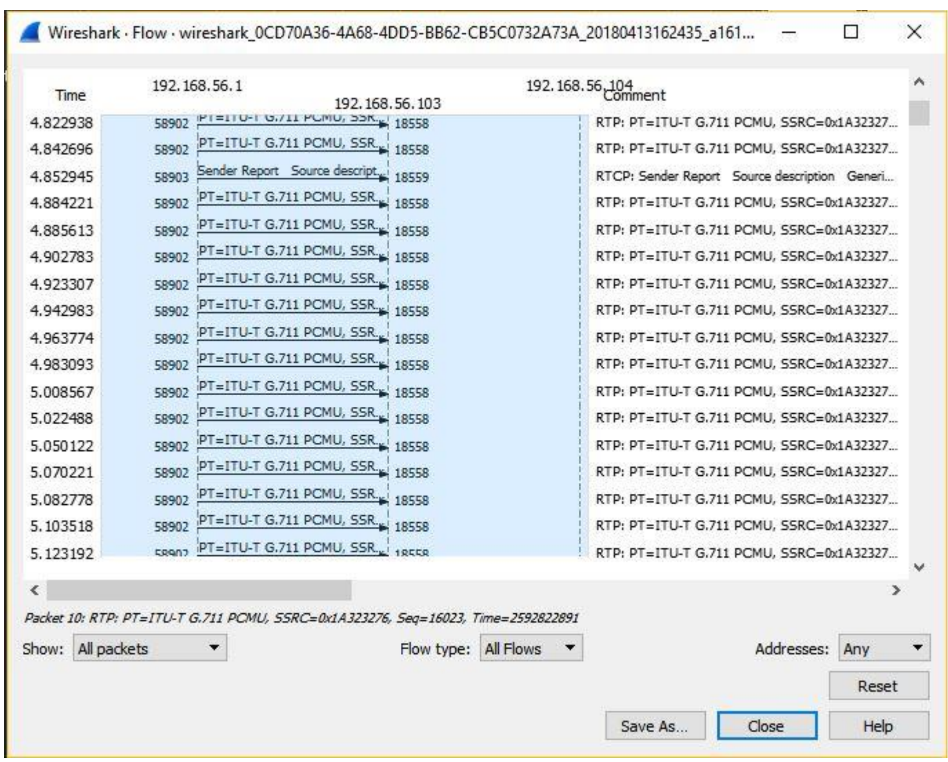


Figure > RTP packets call flow between 2000 and 2010

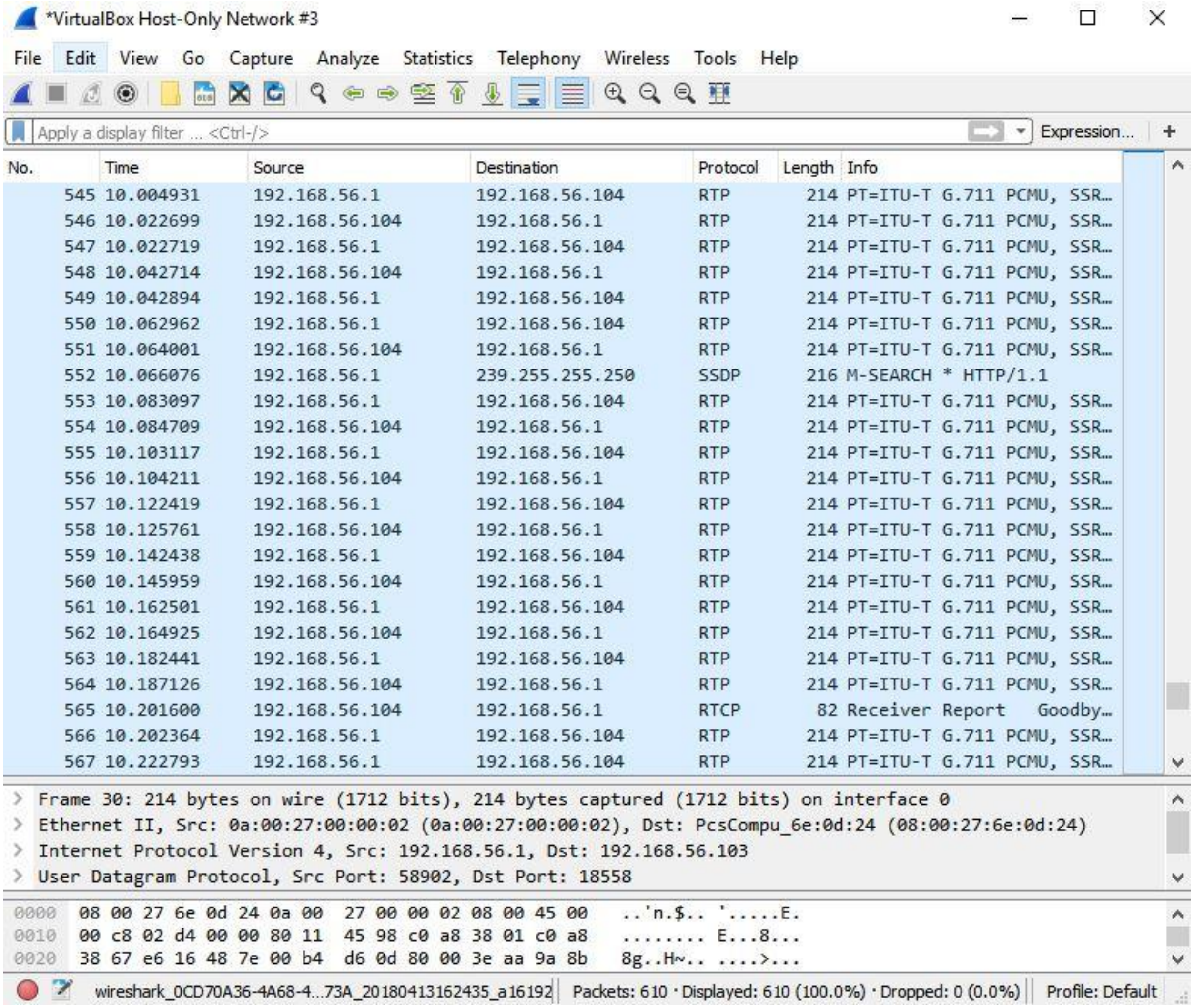


Figure > RTP Packets in the call

3. Phase-1 Summary:

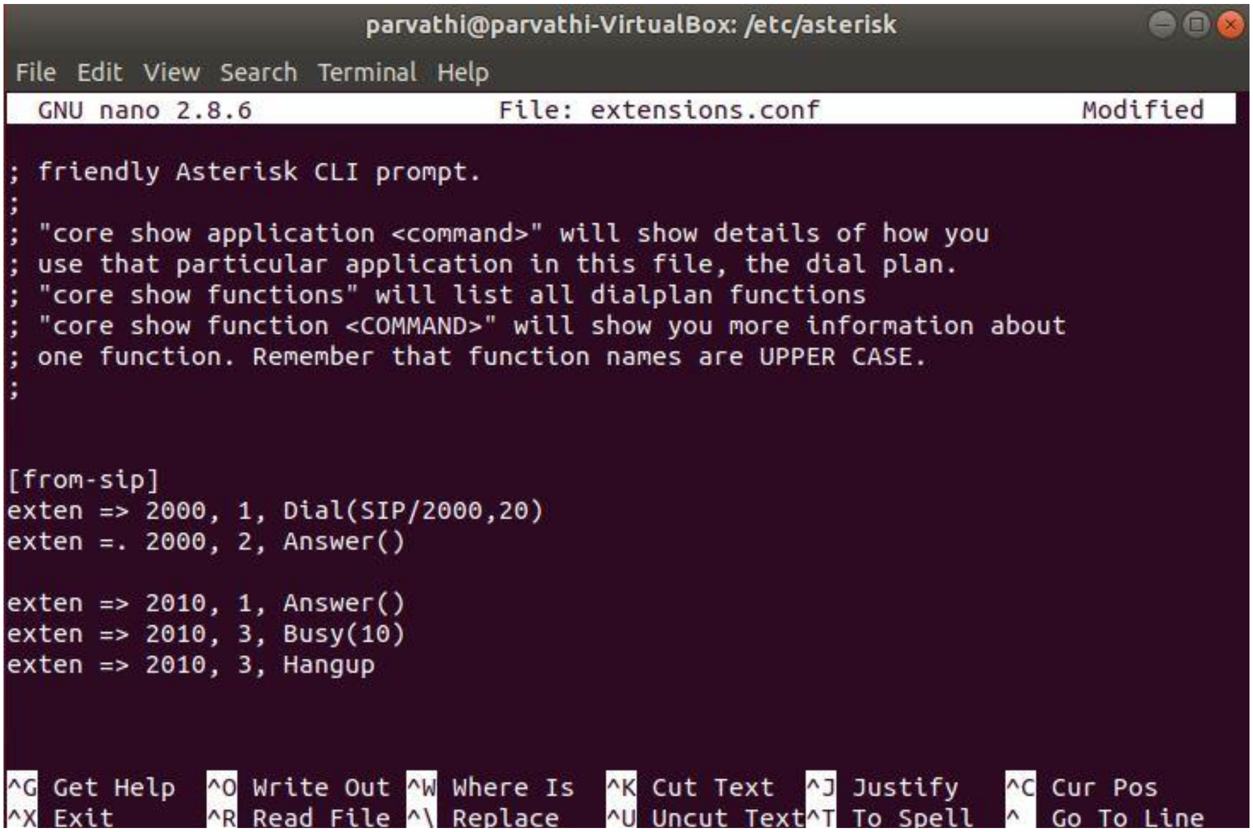
Call established between 2000 and 2010 after invite from 2000.

PHASE 2- BUSY USER:

We set 2010 as a busy user so 2000 would only hear a busy tone when 2010 accepts the call from 2000 user.

1. ## Call establishment.

The sip.conf file is the same as Phase 1. Some changes are made to the extension.conf file for Client 1 and 2. We then give the reload command.



```
parvathi@parvathi-VirtualBox: /etc/asterisk
File Edit View Search Terminal Help
  GNU nano 2.8.6              File: extensions.conf              Modified

; friendly Asterisk CLI prompt.
;
; "core show application <command>" will show details of how you
; use that particular application in this file, the dial plan.
; "core show functions" will list all dialplan functions
; "core show function <COMMAND>" will show you more information about
; one function. Remember that function names are UPPER CASE.
;


[from-sip]
exten => 2000, 1, Dial(SIP/2000,20)
exten =. 2000, 2, Answer()

exten => 2010, 1, Answer()
exten => 2010, 3, Busy(10)
exten => 2010, 3, Hangup



^G Get Help   ^O Write Out  ^W Where Is   ^K Cut Text   ^J Justify    ^C Cur Pos
^X Exit       ^R Read File  ^\ Replace    ^U Uncut Text ^T To Spell   ^  Go To Line
```

2. ## Capture of results

The Wireshark screenshots are attached below:

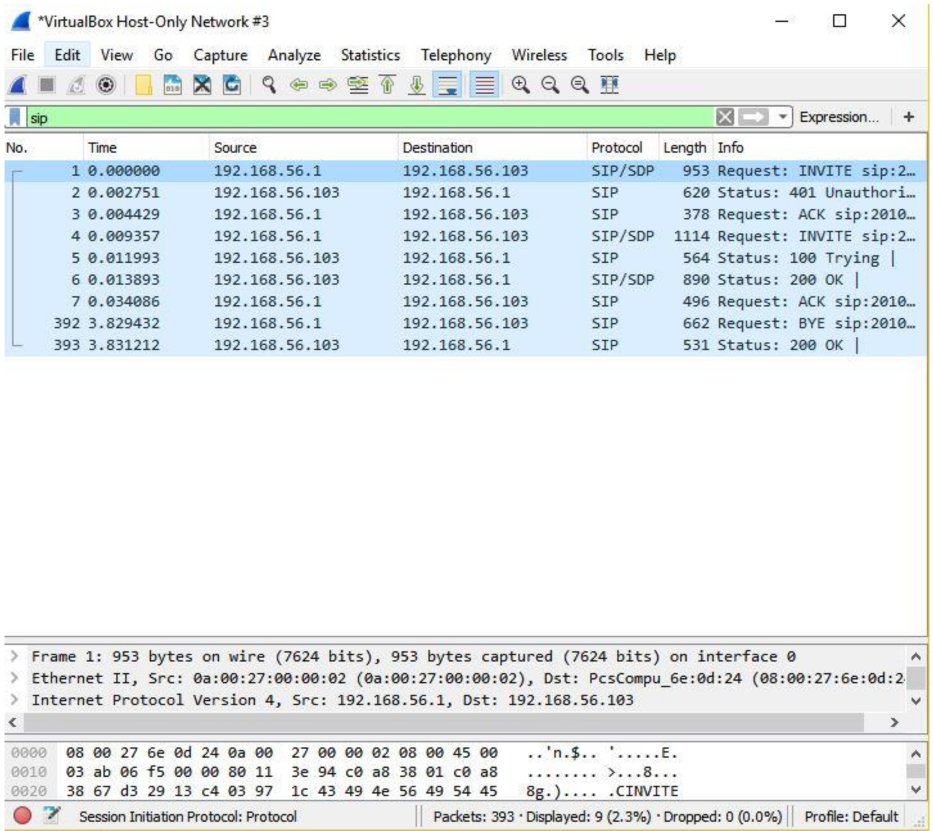


Figure > Call establishment between 2 clients

Above Wireshark capture shows the flow for SIP messages exchanged between the clients.

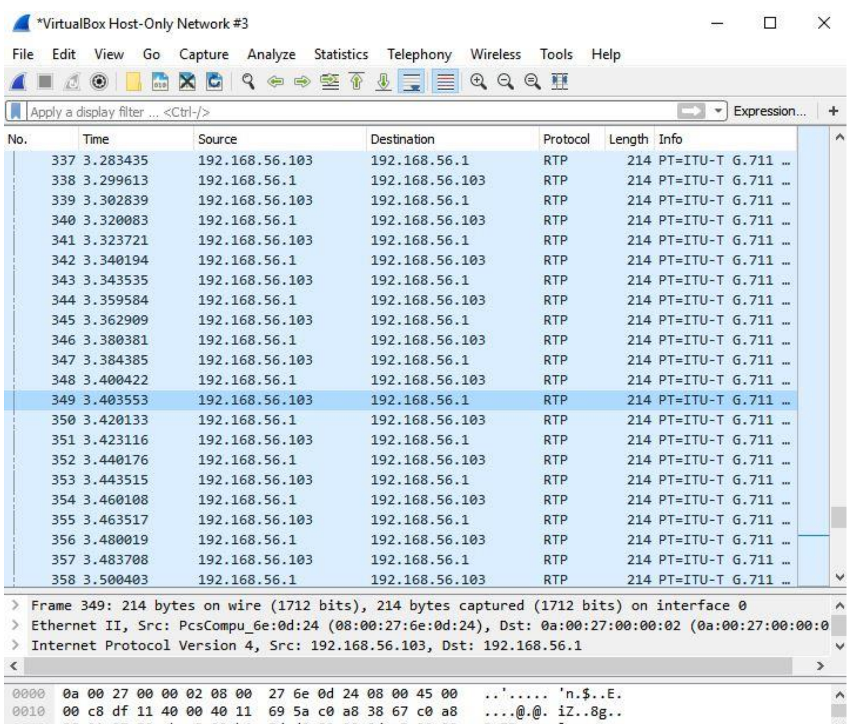


Figure > RTP packets

Above wireshark capture shows the RTP packets exchanged after the call is established between the users.

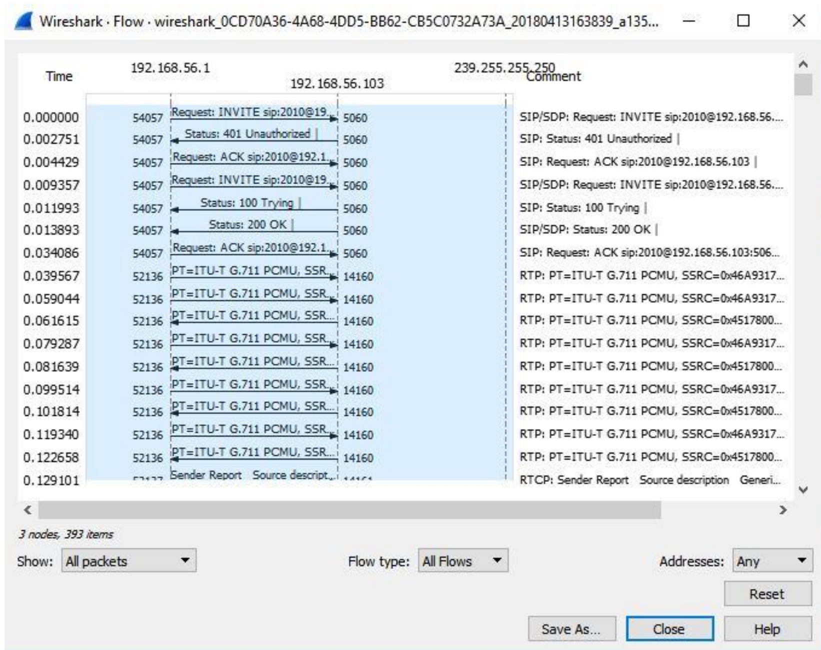


Figure > Packet Flow

Above capture shows the messages exchanged between the clients.

3. Phase-2 Summary:

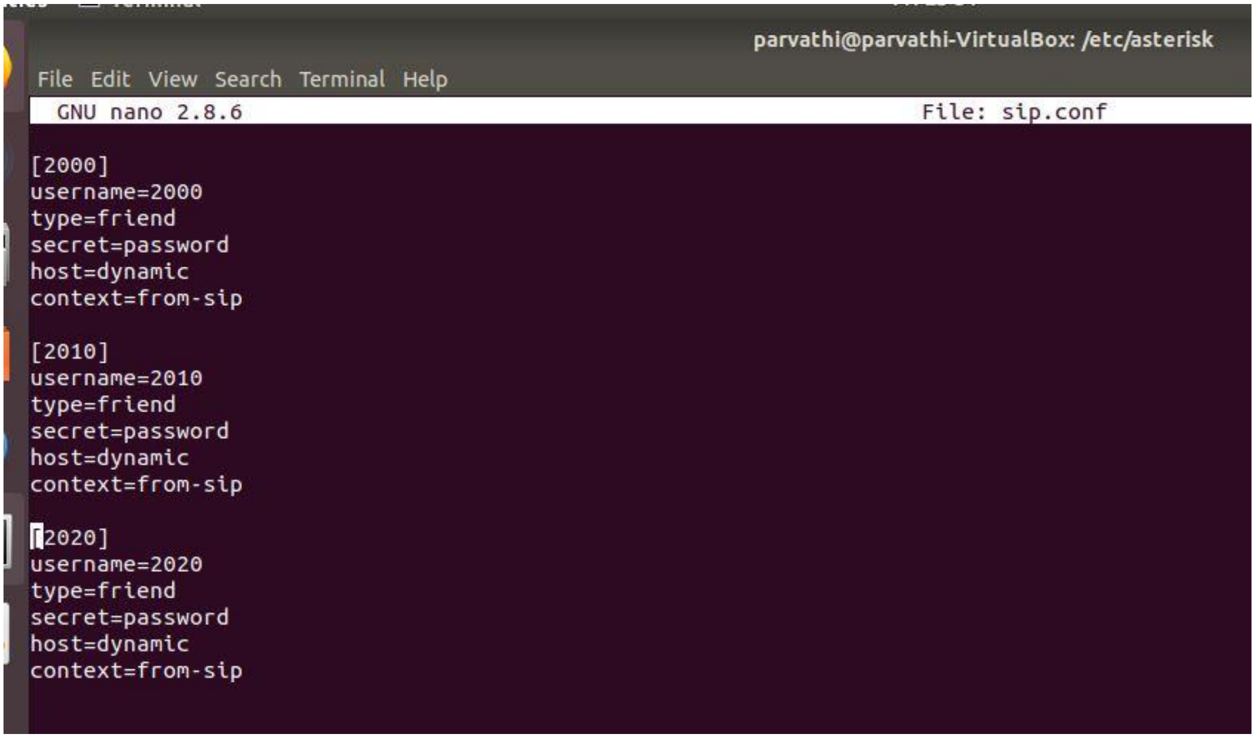
In this phase, 2000 calls 2010 but gets a busy tone.

PHASE 3 - CALL ON HOLD:

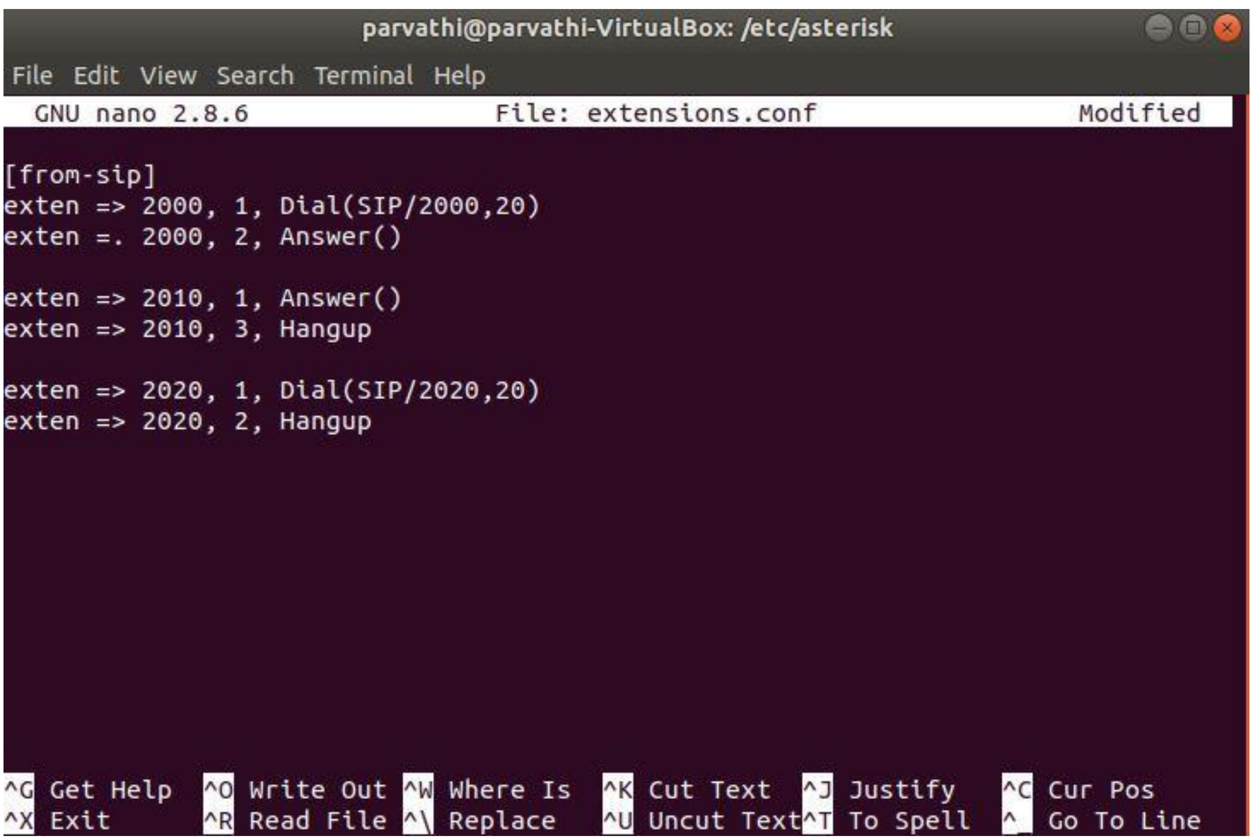
Here, 2000 calls 2010 and establishes a call. 2020 tries to call 2000. Therefore, 2000 user puts 2010 on hold and accepts 2020.

1. Call establishment.

The experiments need three clients. The third client can be included as user 2020 in sip.conf file from Phase I, as follows:



The extensions.conf file is modified and added to the existing values.



2. Capture of results

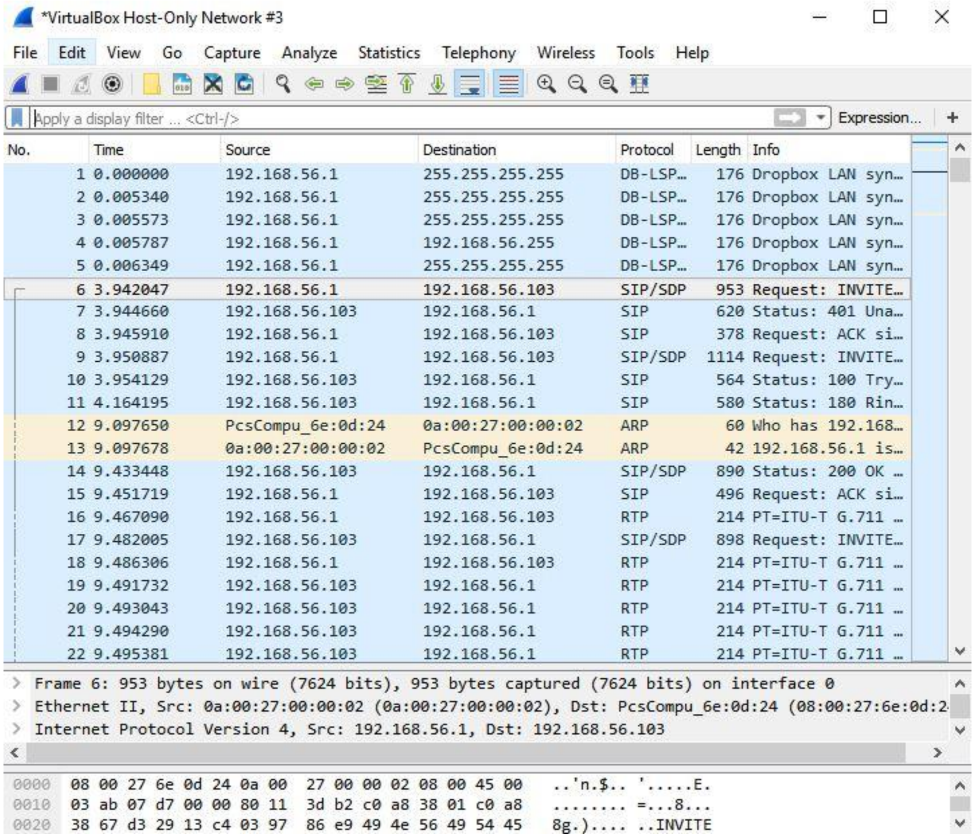


Figure > Calling between two clients

Above Wireshark capture shows the SIP messages between the user 2000 and 2010.

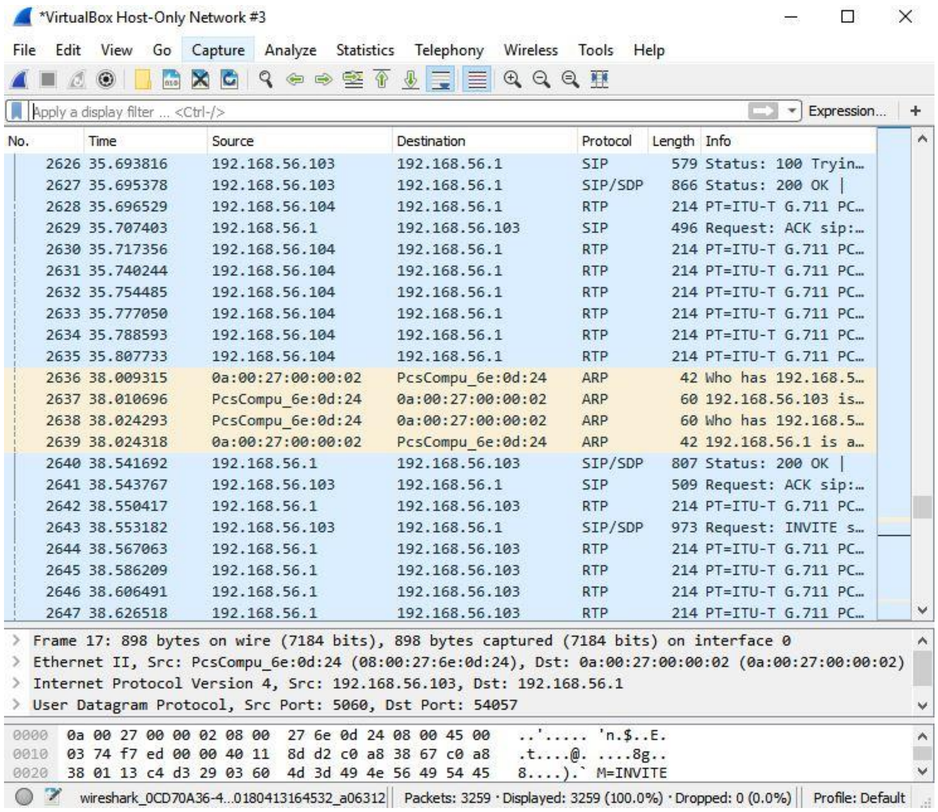


Figure > A Call on HOLD by first user

Above Wireshark screencapture shows the different messages when the 2000 puts the call of 2010 on hold to accept the call of 2020.

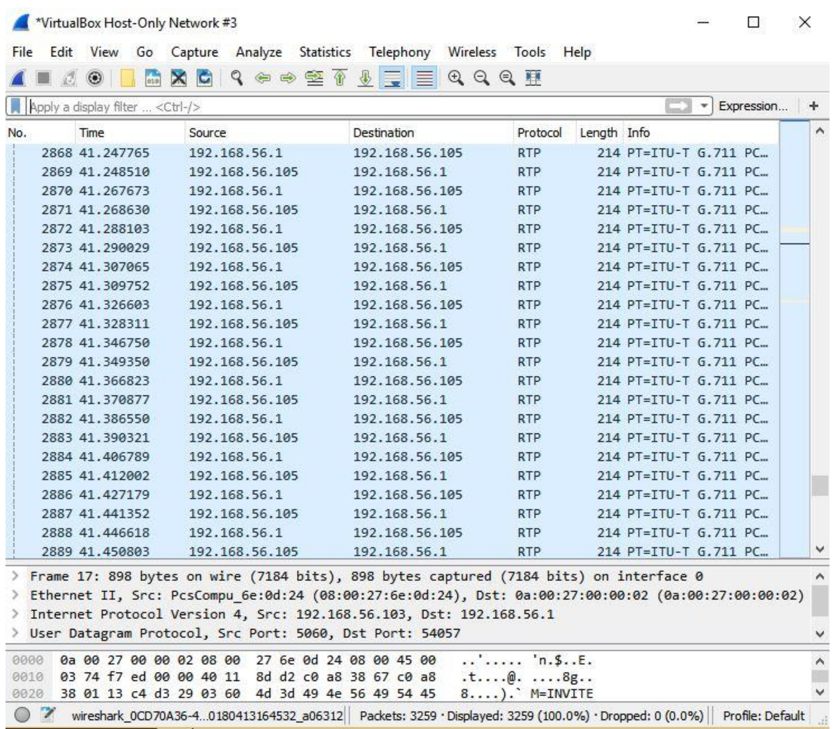


Figure > 2000 accepts 2020 and transmission begins

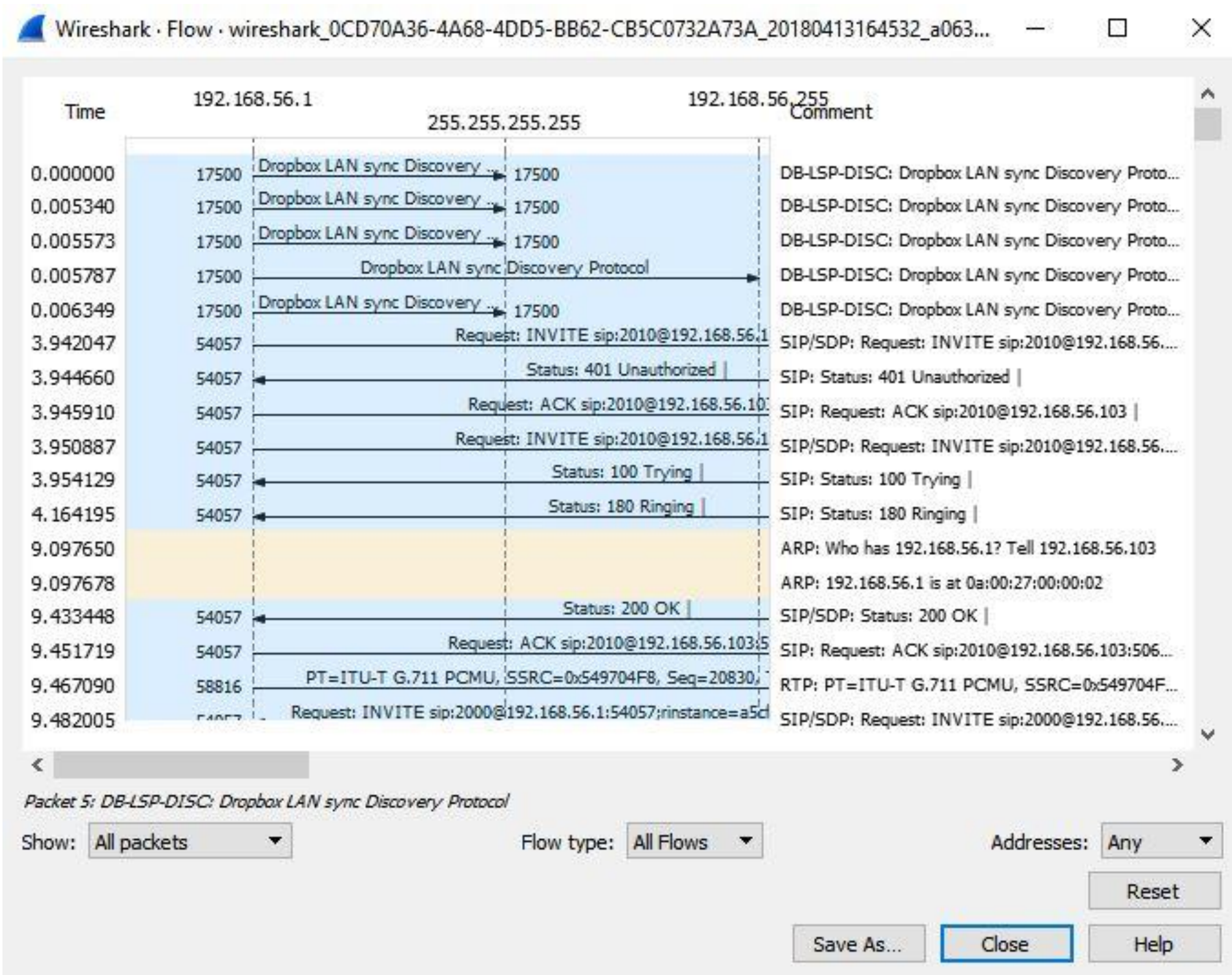


Figure > Call Flow

3. Phase – 3 Summary:

Here, call is already in progress between 2000 and 2010 when 2020 calls. 2000 puts 2010 on hold and accepts 2020. 2010 would hear a hold tone.

PHASE 4 - CALL CONFERENCING:

1. Call establishment.

Here, 2000 and 2010 have an established call. 2020 calls user 2000. 2000 puts 2010 on hold, and accepts the call from 2020. 2000 then merges both the calls together to form a conference call.

2. Capture of results

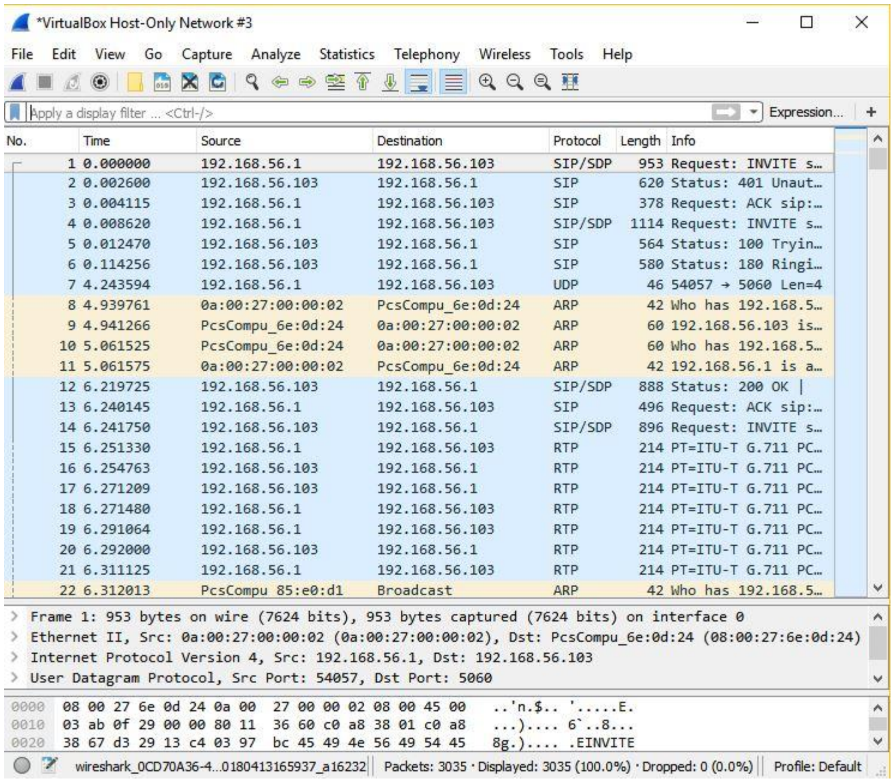


Figure > Call in progress between 2000 and 2010

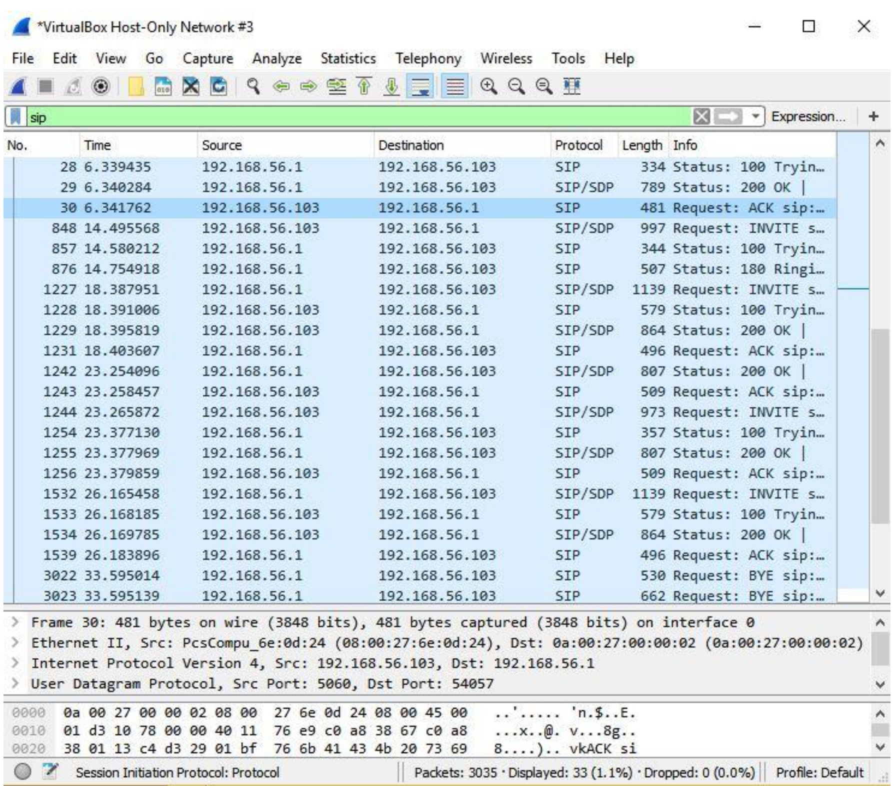


Figure > Invite from 2020

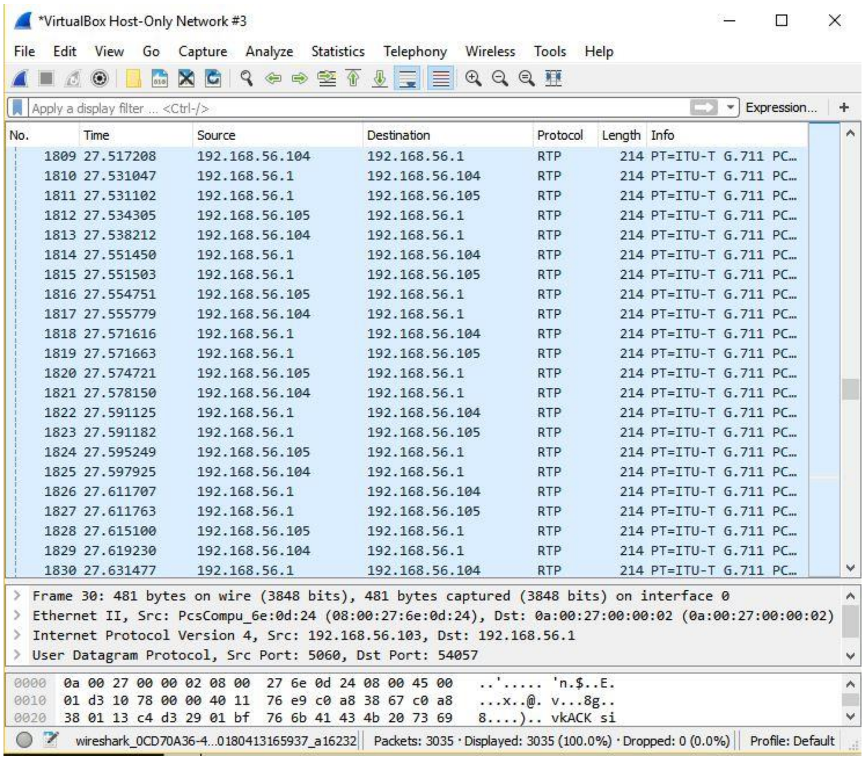


Figure > Call From 2020

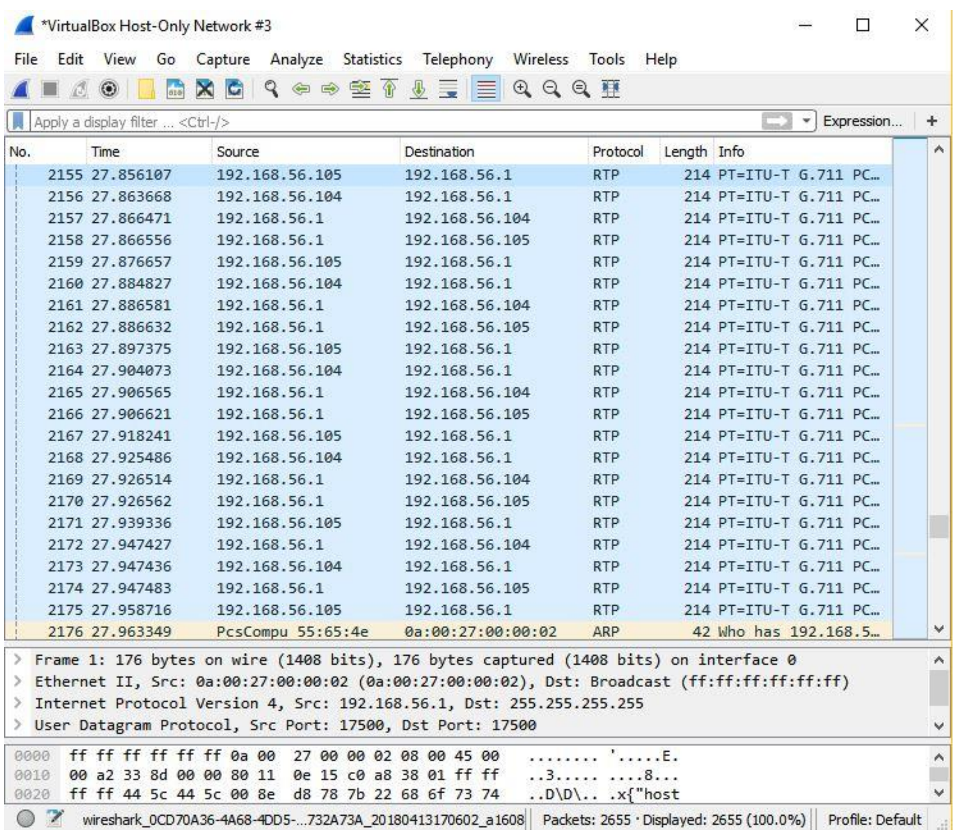


Figure > Merged Calls

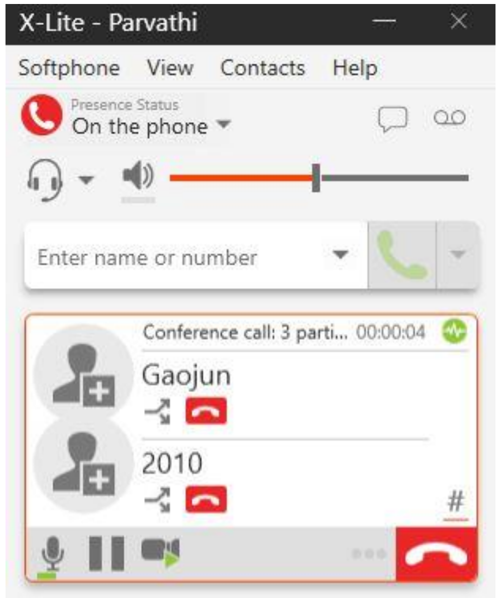


Figure > Merge Calls by client 2000

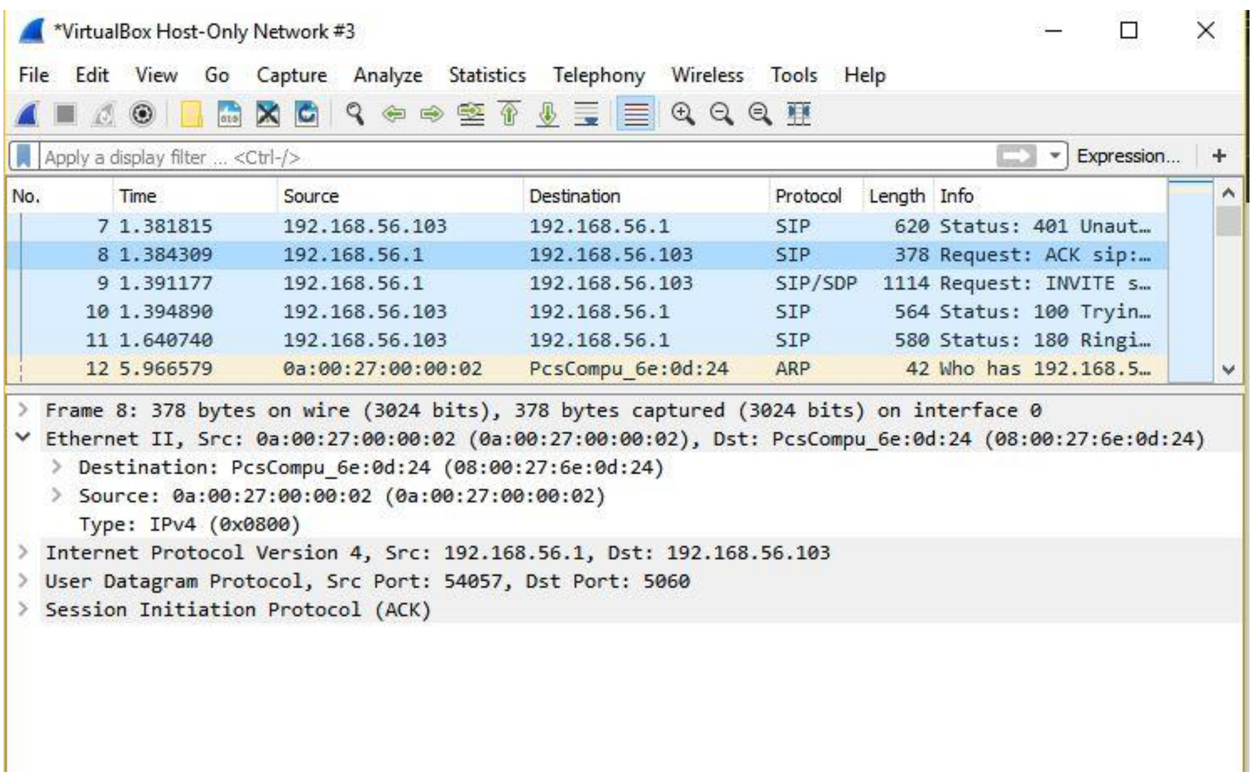


Figure > Media Access Control address of the Server

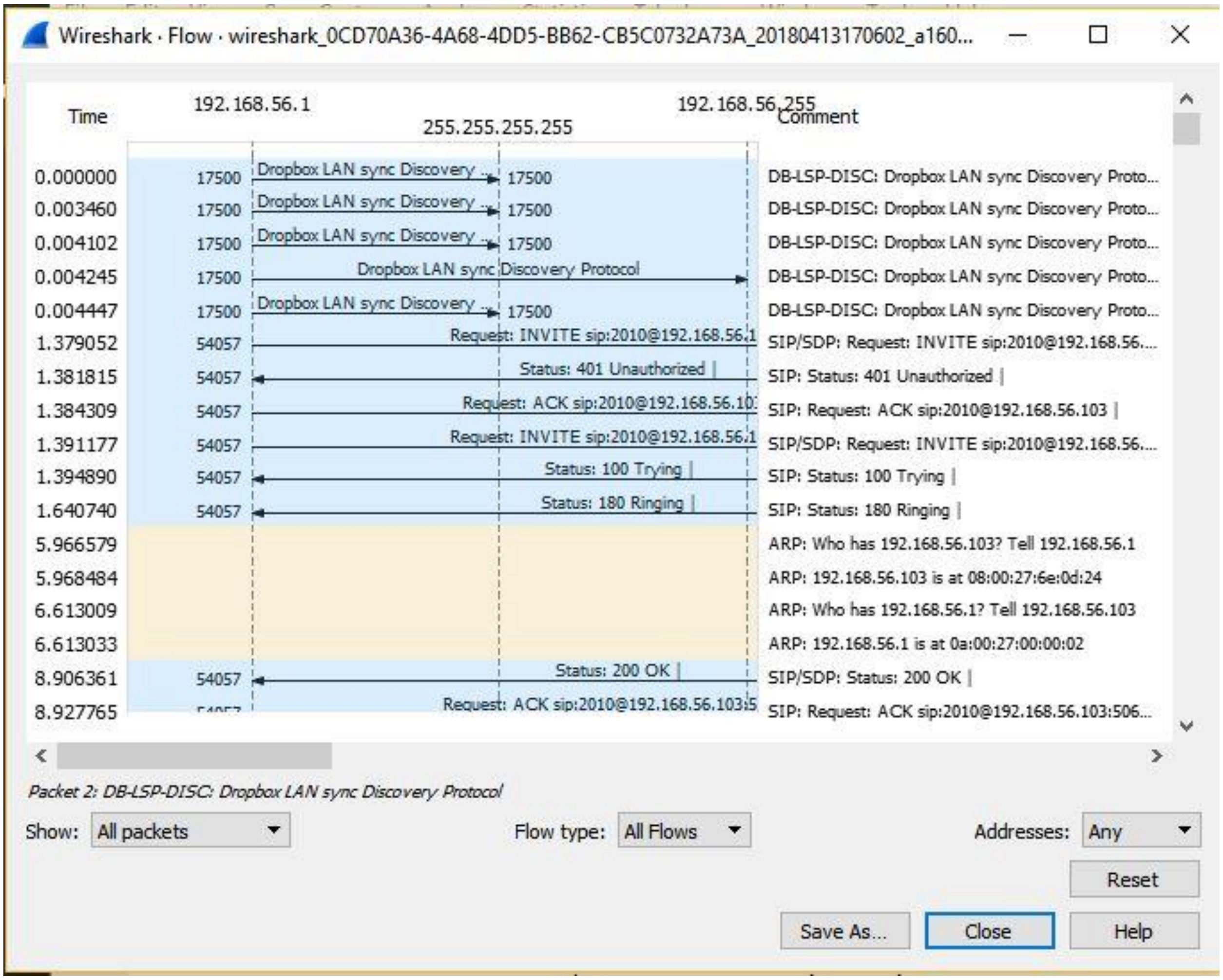


Figure > Call Flow

3. Phase – 4 Summary

Here 2000 and 2010 have a call in progress. 2020 sends a call request connection to 2000. 2000 puts 2010 on hold and accepts 2020. 2000 then invites 2020 to be part of the three user call.

PART-2

Abstract

Part 2 states to create a client using code, with SIP functionalities.

In this part we created a SIP client script using python. The client can successfully register, unregister and send call invite to another client. The working of the client is shown in below figures.

1. **Registering Client:**

This screenshot from the server shows that the client is not registered at the beginning. As seen at the bottom, there is only one client is registered to the server. The registered client is with IP 192.168.56.104 and having user ID 2010.

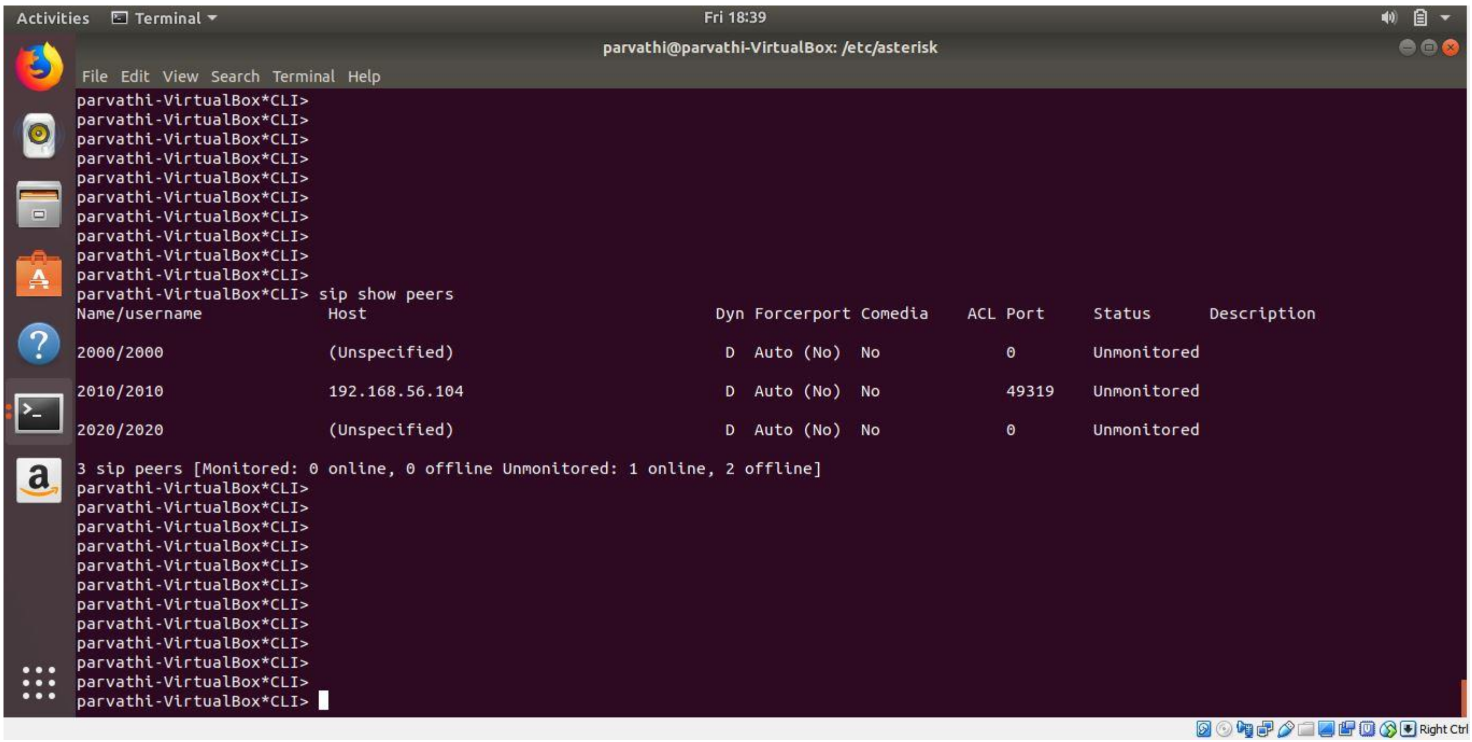


Figure > Only 2010 registered

The client script is now run on an Ubuntu client virtual machine. It requires the client IP [192.168.56.106], Server IP [192.168.56.103], Username [we have specified 2000], Password [which is just password for us].

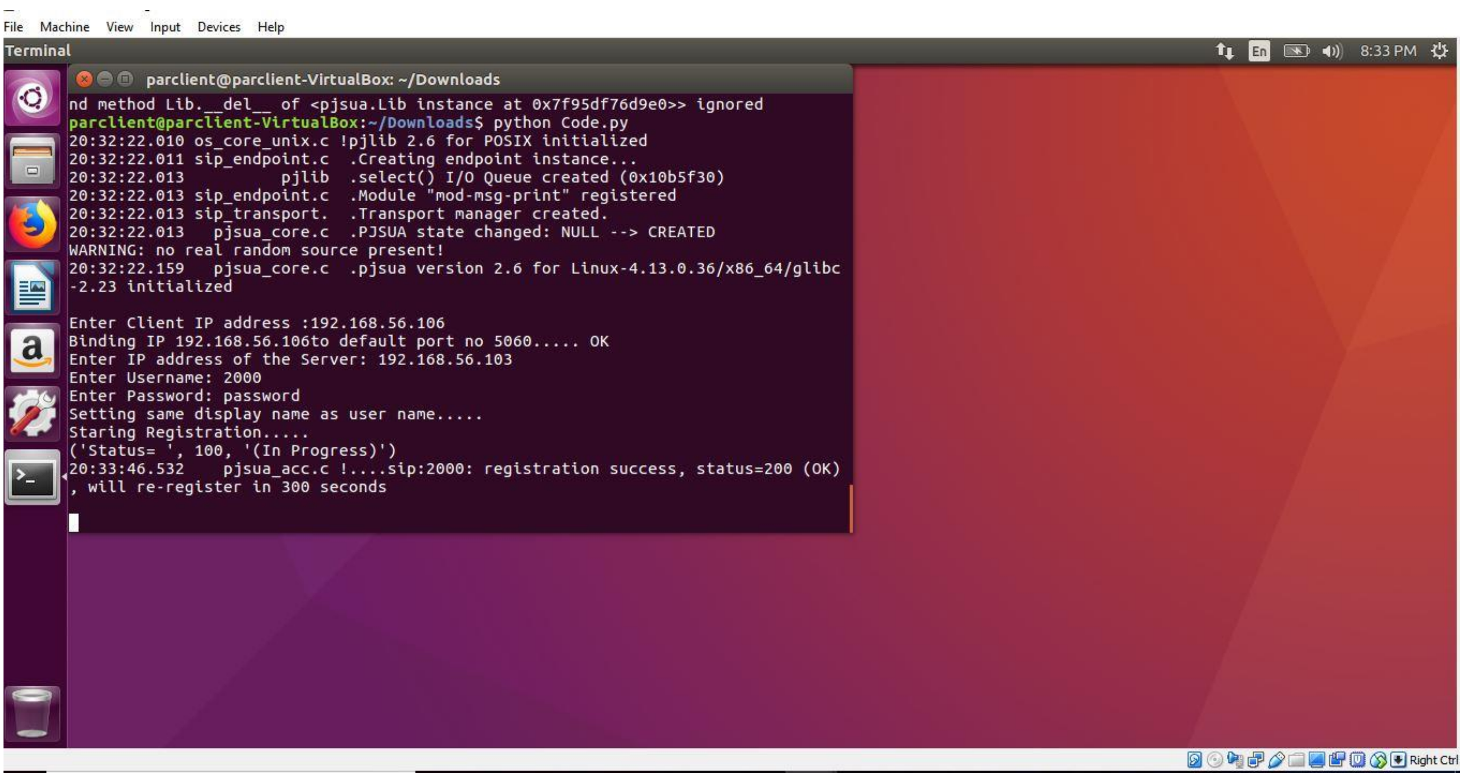


Figure > Client script registration

After the data has been input, the Client script will register itself to the server. The screenshot below shows the server updating the registrations.

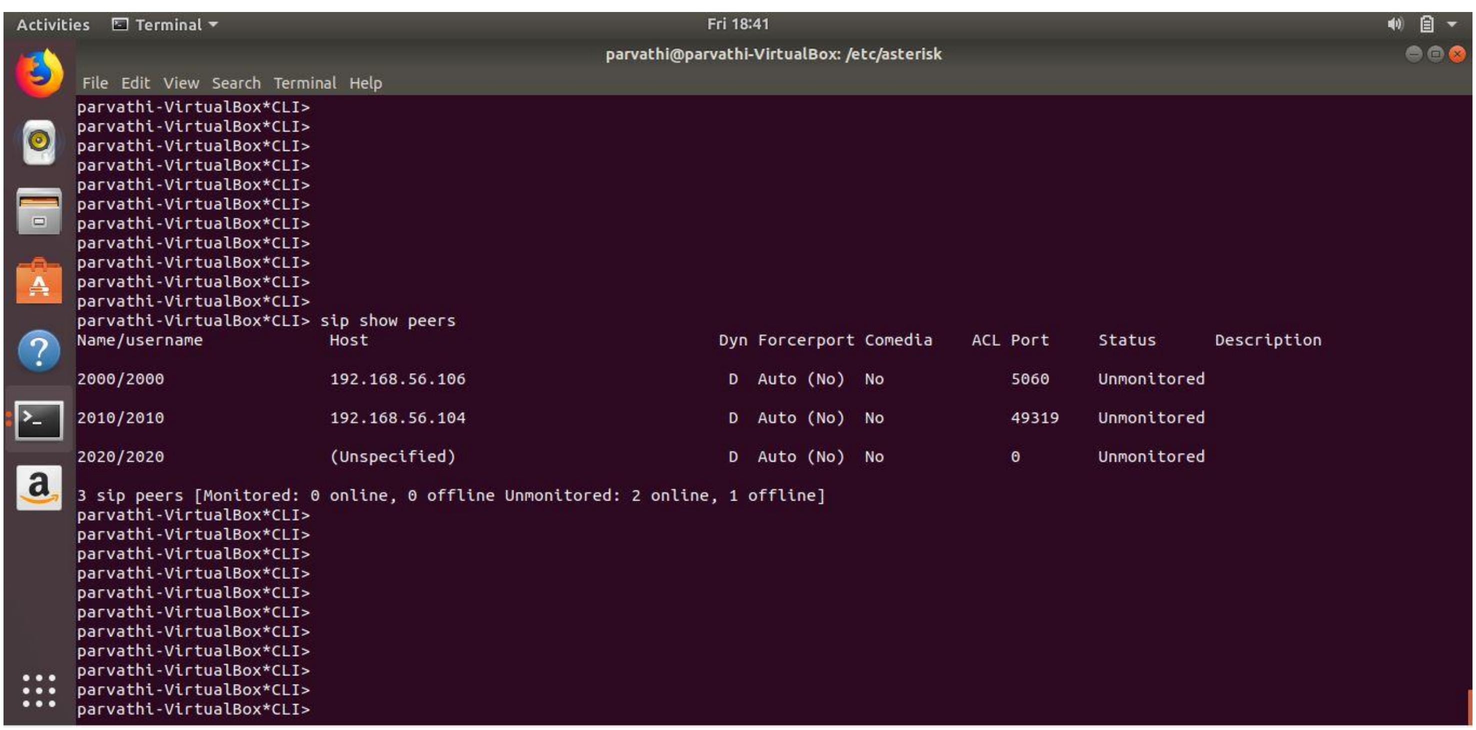
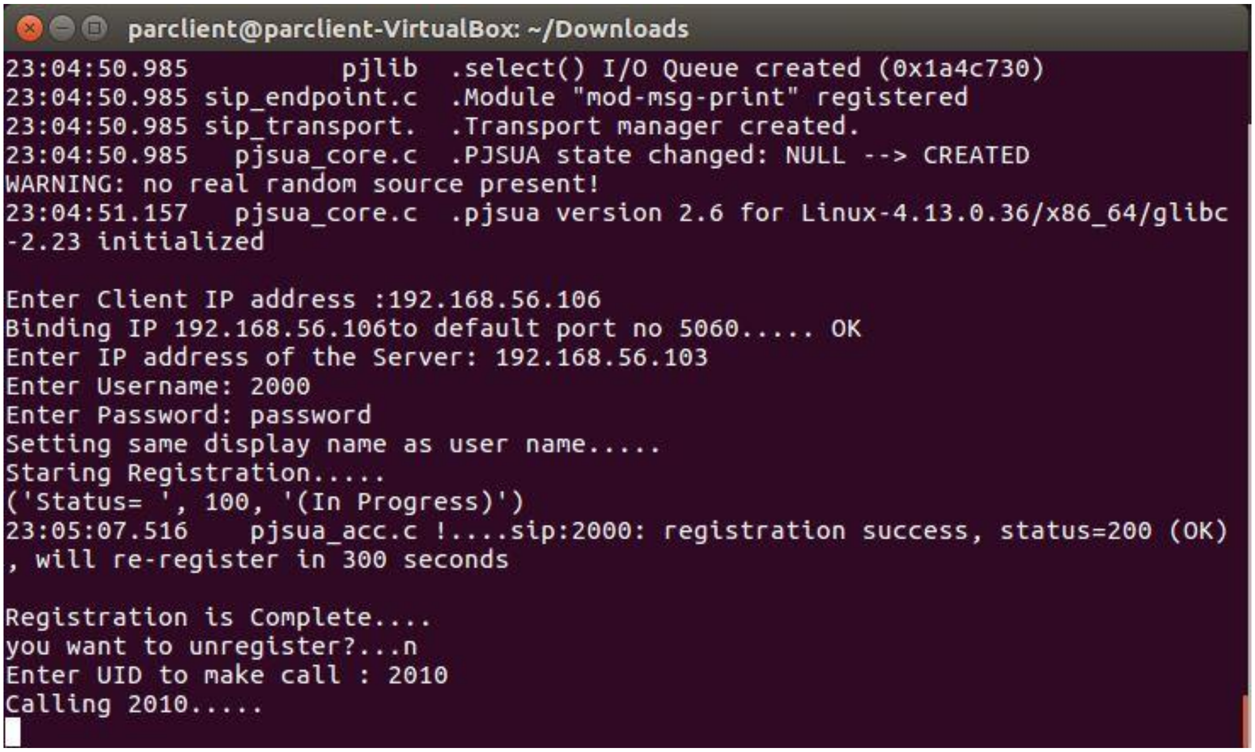


Figure > Registered screenshot from server side

Hence, the registration of client has been completed.

2. Calling:

One the registration process is complete, calling will begin to the user mentioned earlier. [2010].



```
parclient@parclient-VirtualBox: ~/Downloads
23:04:50.985          pjlib  .select() I/O Queue created (0x1a4c730)
23:04:50.985 sip_endpoint.c  .Module "mod-msg-print" registered
23:04:50.985 sip_transport.  .Transport manager created.
23:04:50.985   pjsua_core.c  .PJSUA state changed: NULL --> CREATED
WARNING: no real random source present!
23:04:51.157   pjsua_core.c  .pjsua version 2.6 for Linux-4.13.0.36/x86_64/glibc
-2.23 initialized

Enter Client IP address :192.168.56.106
Binding IP 192.168.56.106to default port no 5060..... OK
Enter IP address of the Server: 192.168.56.103
Enter Username: 2000
Enter Password: password
Setting same display name as user name.....
Staring Registration.....
('Status= ', 100, '(In Progress)')
23:05:07.516    pjsua_acc.c !....sip:2000: registration success, status=200 (OK)
, will re-register in 300 seconds

Registration is Complete....
you want to unregister?...n
Enter UID to make call : 2010
Calling 2010.....
```

Figure > Calling from client script to 2010

With this, the opposite client's X-Lite starts ringing and the other client can answer or decline.

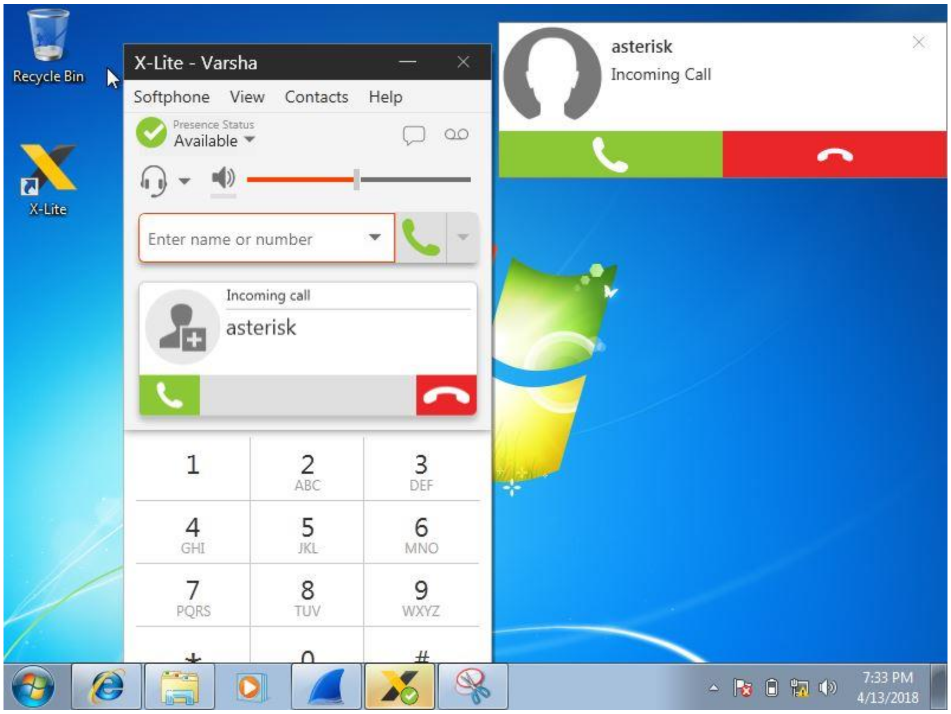


Figure > Incoming call

3. Unregistering Client:

Once the user gives command the script starts unregistering the client from the SIP server. The following figure shows the unregistering process.

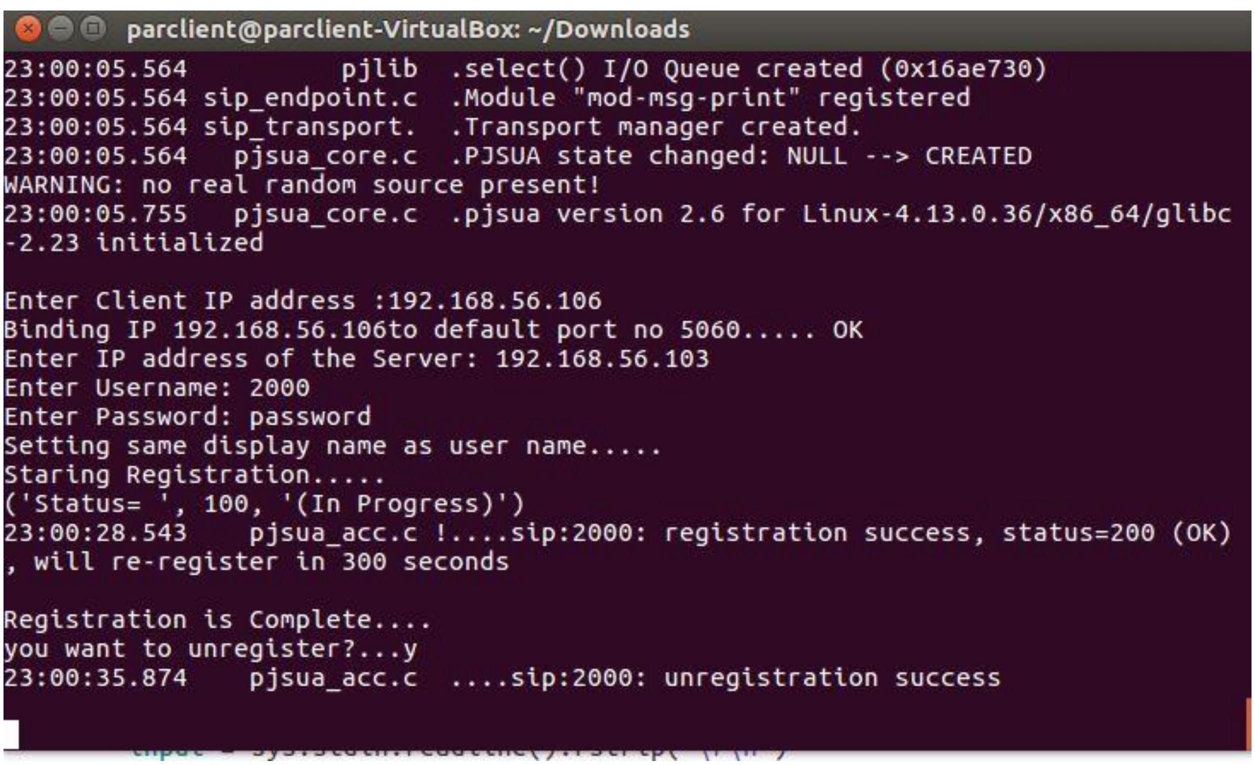


Figure > Unregistration by client script

The above figure shows that the client has successfully been unregistered from the SIP server. The following figure proves the successful registration from the SIP server.

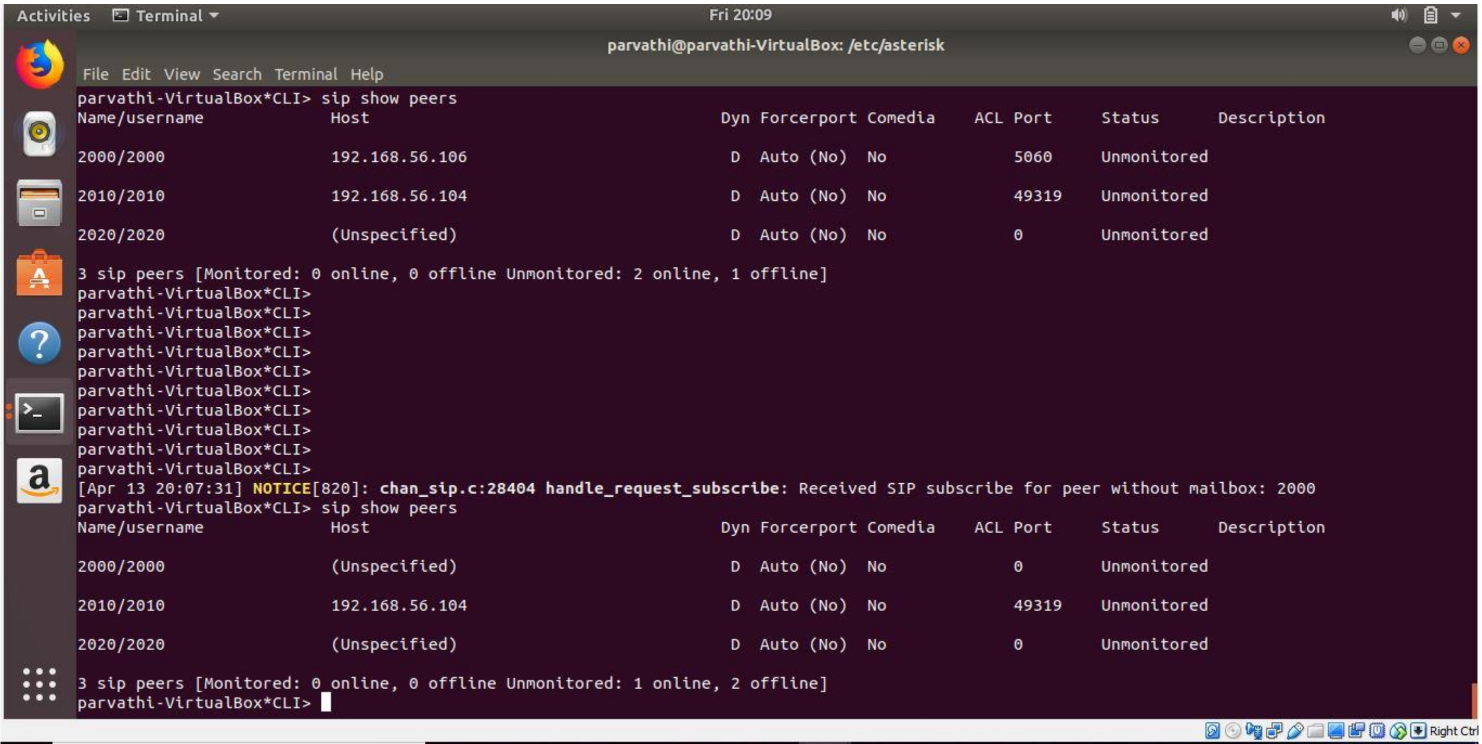


Figure > Shows unregistered client script

Therefore, the client script has successfully unregistered from the Asterisk server.

4. Wireshark Capture:

Below are the Wireshark screenshots showing the SIP invite, register packets from the Python client script. The register packet is intended for the server and the invite packet for the client on the other end.

The SIP registration packet is from the Python script client 192.168.56.106.

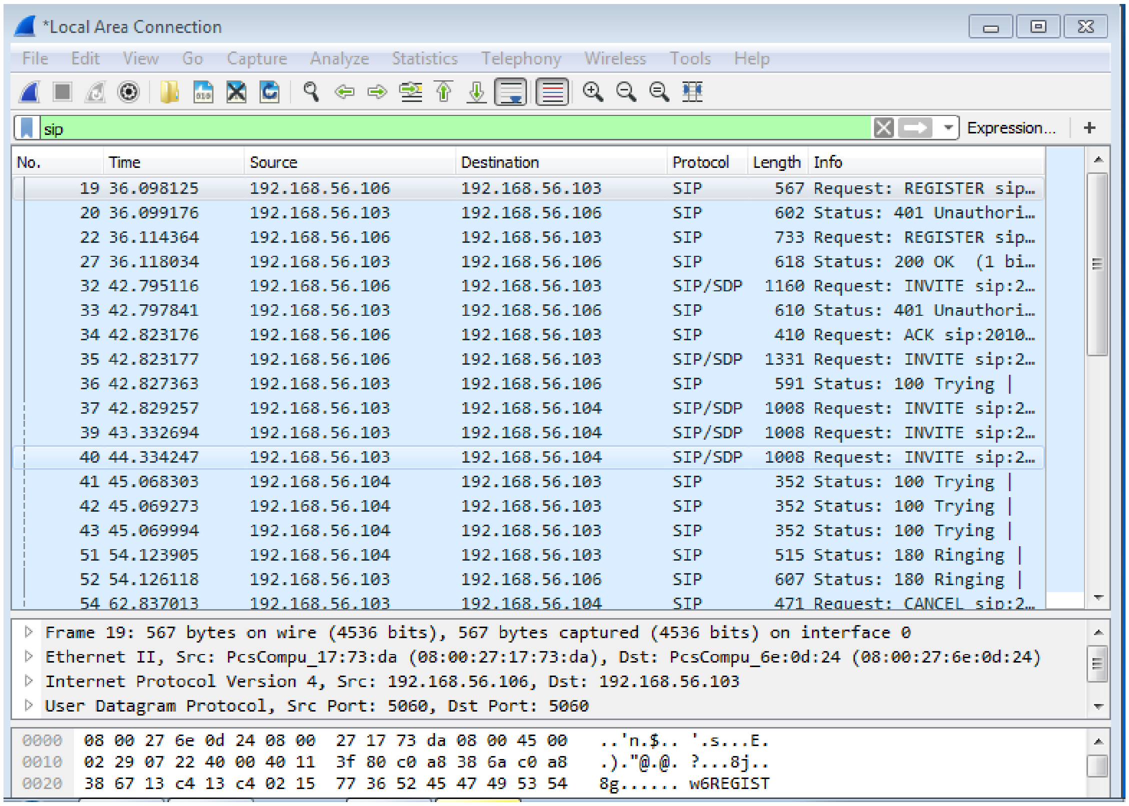


Figure > Call from client script

The ack shows successful call establishment. Below is the screenshot.

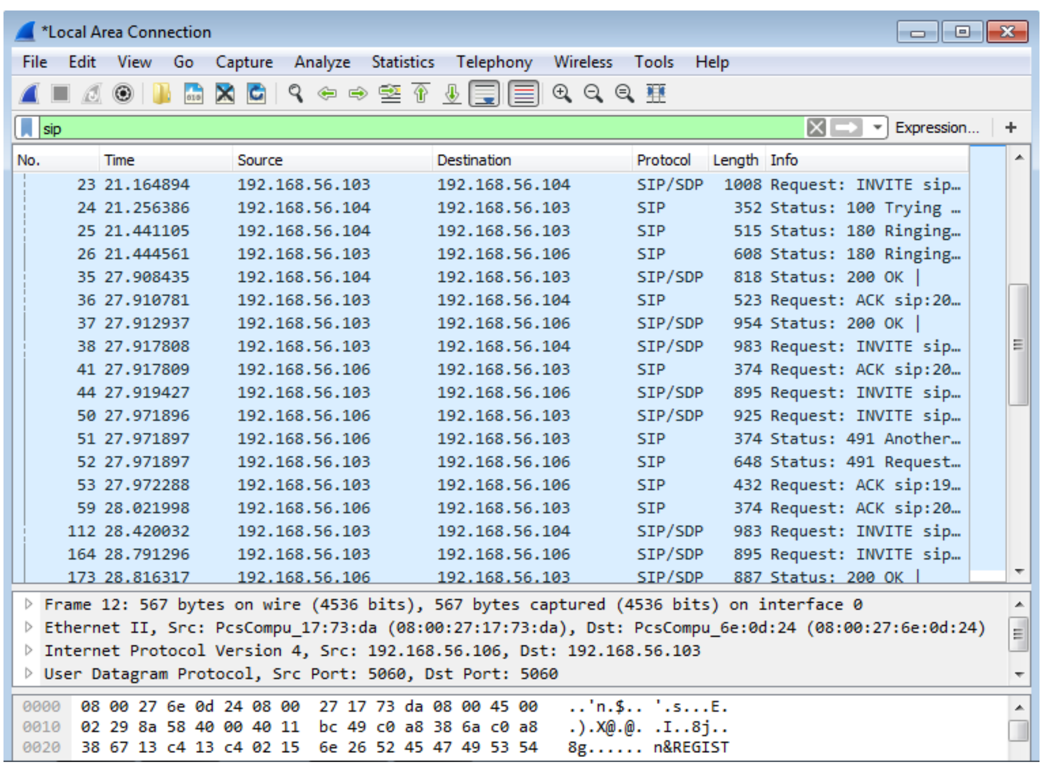
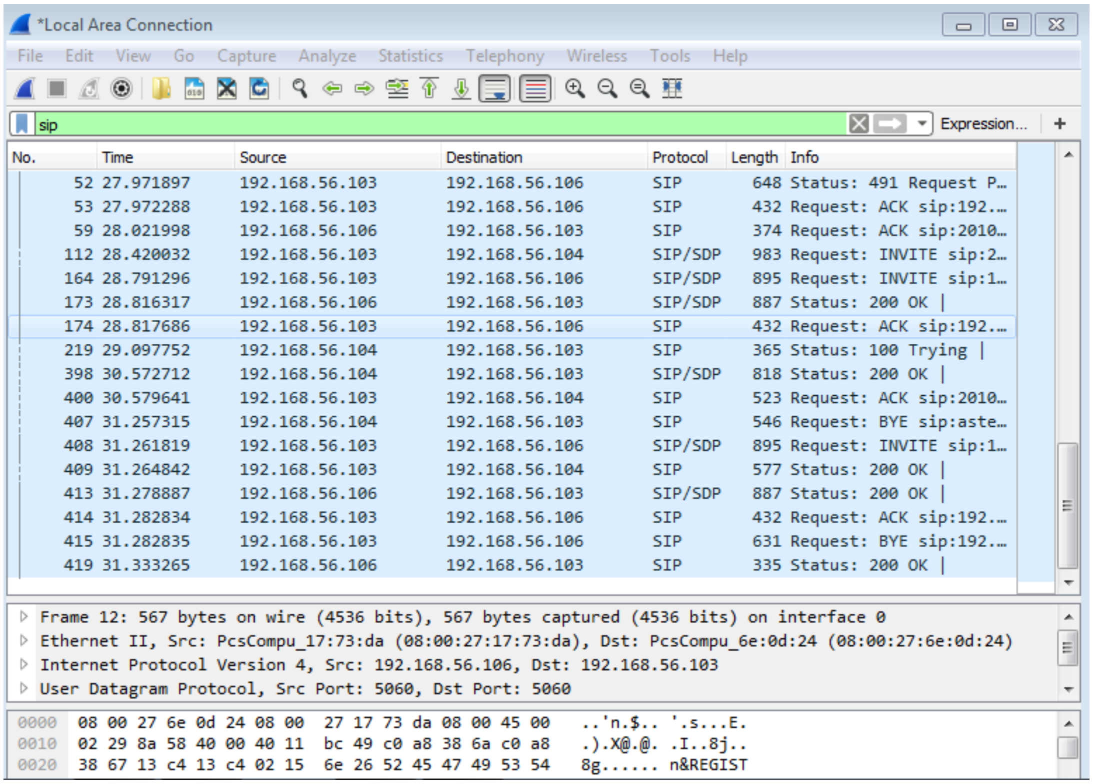


Figure > Call established

The figure below shows the end of the call.



SOURCE CODE:

```
import pjsua as sip
import sys
import time
def log_cb(level, str, len):
	print(str)
#Cll back instance
class Calling_back_account(sip.AccountCallback):
	def __init__ (self,account):
		sip.AccountCallback.__init__(self, account)
	#Clling to URI
class SRCallCallback(sip.CallCallback):
	def __init__(self, call):
		sip.CallCallback.__init__(self, call)
	def Active_state(self):
		print("Call is :", self.call.info().state_text),
		print("last code :", self.call.info().last_code),
		print("(" + self.call.info().last_reason + ")")

		#Disconnection
		if (self.call.info().state_text == 'DISCONNCTD'):
			print 'Press anykey to Unregister.....'
			return

try:
	#Instance creation
	sip_lib = sip.Lib()
	sip_lib.init(log_cfg = sip.LogConfig(level=3, callback=log_cb))
	#creating Transport Object Instance...
```

```
client_IP_address = raw_input('Enter Client IP address :')
transport_socket = sip.TransportConfig()
transport_socket.port = 5060
transport_socket.bound_addr = client_IP_address
print('Binding IP ' + client_IP_address + 'to default port no 5060.....'),
transport_bind =sip_lib.create_transport(sip.TransportType.UDP,transport_socket)
print('OK')
#Starting SIP libraries
sip_lib.start()
sip_lib.set_null_snd_dev()
#Starting Registering Process
r_IP=raw_input("Enter IP address of the Server: ")
r_name=raw_input("Enter Username: ")
r_pwd=raw_input("Enter Password: ")
print 'Setting same display name as user name.....'
r_Dname=r_name
print 'Staring Registration.....'
conf_of_account = sip.AccountConfig(domain = r_IP, username = r_name, password=r_pwd, display =
r_Dname, proxy = 'sip:%s:5060' % r_IP)
conf_of_account.id ="sip:%s" % (r_name)
conf_of_account.reg_uri ='sip:%s:%s' % (r_IP,transport_socket.port)
account_callback = Calling_back_account(conf_of_account)
acc = sip_lib.create_account(conf_of_account,cb=account_callback)
#Setting value to Calling_back_account class
acc.set_callback(account_callback)
print('Status= ',acc.info().reg_status,'(' + acc.info().reg_reason + ')')
time.sleep(5)
print 'Registration is Complete....'
Reg_unreg=raw_input("you want to unregister?...")
if (Reg_unreg=="y"):
```

```
            acc.set_registration(False)

        else:

            #Cll
            c_ID = raw_input('Enter UID to make call : ')
            print 'Calling %s.....' % (c_ID)
            s_URI = 'sip:%s@%s:%s' % (c_ID,r_IP,transport_socket.port)
            call = acc.make_call(s_URI, SRCallCallback(acc))

        #Unreg
        input = sys.stdin.readline().rstrip('\r\n')
        print 'Unregistering.....'
        time.sleep(2)
        sip_lib.destroy()
        time.sleep(2)
        sip_lib= None
        sys.exit(1)

    #Excpt
    except sip.Error, err:
        print 'Initializations Error', err
        sip_lib.destroy()
```

Conclusion:

In the project we gained practical experience for configuration of the server and clients. We also observed the signaling messages in SIP. The first part included different call scenarios between X-Lite clients registered to Asterisk server. In Part-2, we installed PJSIP library to create a client using a Python script.

We implemented four different call scenarios while observing the SIP messages sent by capturing screenshots of packets in Wireshark. We also studied the RTP messages transmitted. This project is very helpful for us in understanding how IP-telephony works using SIP and RTP protocol.

References:

[1] http://www.asteriskguru.com/tutorials/

[2] http://www.pjsip.org/

[3] RFC 3261, *SIP: Session Initiation Protocol*

[4] http://www.voip-info.org/wiki/view/PJSIP