Implementation of SIP URI Service Discovery

Akshat Sikarwar
COMS E6901 - Fall 2008
Department of Electrical Engineering
Columbia University
New York, USA - 10027

as3515@columbia.edu

ABSTRACT

This project report documents the implementation of the Internet-Draft (I-D) *SIP URI Service Discovery using DNS-SD* [1] for SIP Communicator [2].

Keywords

SIP, Bonjour, Zeroconf, service discovery, SIP Communicator

1. INTRODUCTION

Using the mechanisms described in the I-D, a SIP user agent (UA) can communicate with another UA even when no SIP registrar is available. SIP Communicator (SC) was chosen as the target for the implementation. The soft-phone was modified such that it advertises and discovers UAs and makes calls to the discovered contacts.

Consider two users, Alice and Bob, who are running SIP UAs on their laptop computers on the same subnet. Assume that a DNS-SD/mDNS implementation [3], such as Bonjour or Avahi, is installed and running on both computers. Further assume that SIP registrar and proxy server are not available to the UAs (there is no SIP server in the network or the UAs are not configured with the local servers, for instance.) Alice, knowing that Bob is in the vicinity with his computer connected probably to the same subnet as hers, would like to make a SIP call to Bob.

If the UAs use the mechanism in the I-D, this requirement can be met.

The UAs (or contacts) discovered through this mechanism will be referred to as *SIP Bonjour* contacts in this document.

2. BACKGROUND

SC is an open source, Java based VoIP soft-phone. It is also a FSF High Priority Project [4] and is under active development. At time of writing this document, version1.0-alpha3 was under development.

SC uses Apache Felix [5] as its application framework. Felix is an implementation of OSGi [6] by the Apache Foundation. For this implementation, modifications to the SIP protocol bundle were required. A bundle is like a module or a plug-in in OSGi terminology.

The SIP Protocol bundle was extended to advertise the SIP account using Zeroconf, display the discovered UAs in a separate group in the buddy list and make calls to them even if SC was offline. Mechanisms were put in such that if SC was online (as in, logged into a SIP proxy server) and had other SIP contacts in the buddy list, they were called via the configured proxy server, while the contacts discovered through Zeroconf were called directly (p2p).

JmDNS was chosen as the Zeroconf implementation in this project. JmDNS is a pure Java implementation of multi-cast DNS and can be used for service registration and discovery in local area networks. JmDNS is fully compatible with Apple's Bonjour. [7]

SC uses JAIN SIP (or JSIP) as its SIP stack. JSIP is freely available software, developed at the National Institute of Standards and Technology (NIST), an agency of the US Federal Government. [8]

3. IMPLEMENTATION

3.1 SIP URI Advertisement Format

The Zeroconf service advertisement for SIP URI is of the following form:

```
<Instance> . <Service> . <Domain>
```

`<Instance>` is the SIP URI of the UA.

`<Service>` is one of "`_sipuri._udp`", "`_sipuri._tcp`", or "`_sipuri._sctp`", depending on the transport protocol desired by the UA.

`<Domain>` specifies the DNS sub-domain where the service instance is registered. It may be "`local.`", indicating the mDNS local domain, or it may be a conventional domain name such as "`example.com.`".

In addition, the TXT record of the advertisement may contain any of the following:

- `txtvers`
- `name`
- `contact`

Their meaning is self-explanatory.

3.2 General Design

Whenever the UA discovers a contact, it verifies if the contact is already in the contact list. This is required because the service advertisements are usually received multiple times.

This service announcement contains all the necessary information that is required to make a SIP call to the contact.

Similarly, when a contact terminates, the notification is received via Zeroconf and the contact is removed from the buddy list.

Before a UA terminates, it must remove its Zeroconf advertisement, so that this event (of its closing down) is sent to all the interested parties.

All of these functions are implemented in `BonjourService` class in the SIP protocol bundle. At this time the implementation supports communication over UDP only. The *name* and *contact* TXT records are included as part of the Zeroconf advertisement.

In addition, this class also provides a kind of look-up service (whether a contact was discovered using Zeroconf or not) to SC. This is required, as this determines whether the call is made via a proxy server or directly to UA.

A new SIP stack is instantiated in the `OperationSetBasicTelephonySipImpl` class, which takes care of making the direct p2p SIP calls to SIP Bonjour contacts.

3.3 Code Organization

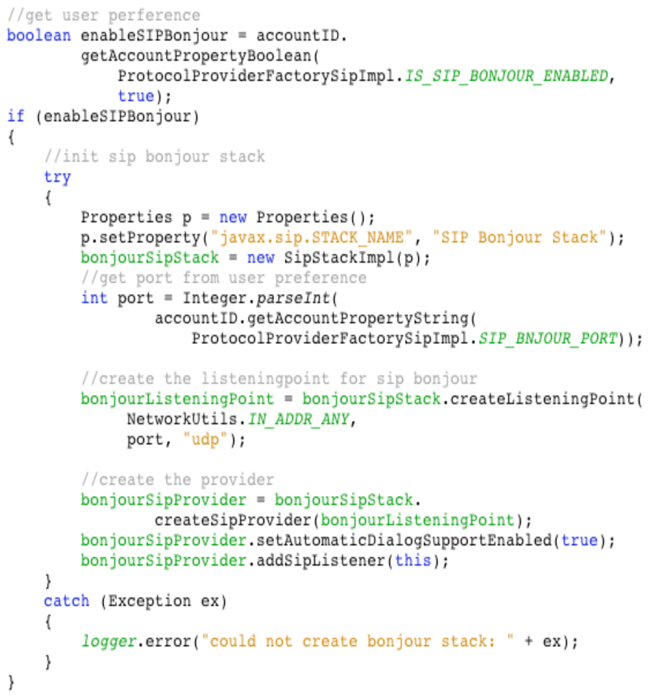
The `BonjourService` class is responsible for handling the Zeroconf events.

A new SipProvider and ListeningPoint was created to respond to SIP events and to make calls to SIP Bonjour contacts. Additional ListeningPoint could not be added to the existing SipProvider because this is prohibited in the JAIN SIP implementation. Following is from the javadoc for `addSipListener` method of `SipProvider` (JAIN SIP):

If there is a ListeningPoint with the same transport but different IP or port, the implementation is expected to throw an exception.

By default a UDP ListeningPoint is created for the proxy server connection, which listens on port 5060. Therefore, we need a new SipProvider with required UDP ListeningPoint. Since it is not possible to have multiple ListeningPoints bound to the same port, the ListeningPoint for SIP Bonjour binds to UDP port 5070.

Following code snippet shows the initialization of the SIP Bonjour SIP stack:



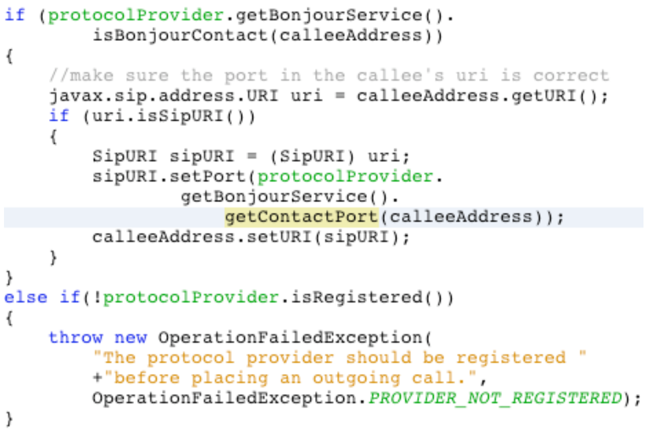
```
//get user perference
boolean enableSIPBonjour = accountID.
        getAccountPropertyBoolean(
            ProtocolProviderFactorySipImpl.IS_SIP_BONJOUR_ENABLED,
            true);
if (enableSIPBonjour)
{
    //init sip bonjour stack
    try
    {
        Properties p = new Properties();
        p.setProperty("javax.sip.STACK_NAME", "SIP Bonjour Stack");
        bonjourSipStack = new SipStackImpl(p);
        //get port from user preference
        int port = Integer.parseInt(
                accountID.getAccountPropertyString(
                    ProtocolProviderFactorySipImpl.SIP_BNJOUR_PORT));

        //create the listeningpoint for sip bonjour
        bonjourListeningPoint = bonjourSipStack.createListeningPoint(
             NetworkUtils.IN_ADDR_ANY,
             port, "udp");

        //create the provider
        bonjourSipProvider = bonjourSipStack.
                createSipProvider(bonjourListeningPoint);
        bonjourSipProvider.setAutomaticDialogSupportEnabled(true);
        bonjourSipProvider.addSipListener(this);
    }
    catch (Exception ex)
    {
        logger.error("could not create bonjour stack: " + ex);
    }
}
```

Figure 1 Initialize SIP stack

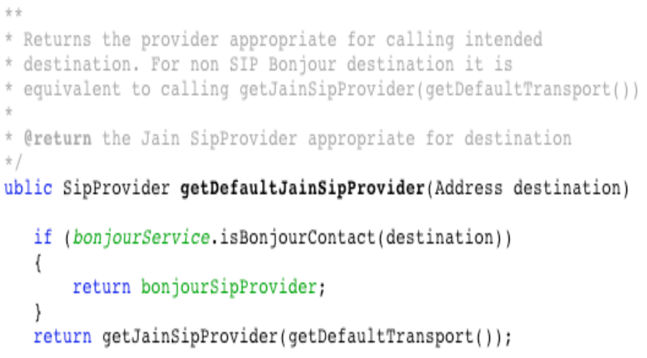
At the time of placing an outgoing call, the check for registered protocol provider needs to be modified so sip contacts can be called even if registration has not happened.



```
if (protocolProvider.getBonjourService().
        isBonjourContact(calleeAddress))
{
    //make sure the port in the callee's uri is correct
    javax.sip.address.URI uri = calleeAddress.getURI();
    if (uri.isSipURI())
    {
        SipURI sipURI = (SipURI) uri;
        sipURI.setPort(protocolProvider.
                getBonjourService().
                        getContactPort(calleeAddress));
        calleeAddress.setURI(sipURI);
    }
}
else if(!protocolProvider.isRegistered())
{
    throw new OperationFailedException(
        "The protocol provider should be registered "
        +"before placing an outgoing call.",
        OperationFailedException.PROVIDER_NOT_REGISTERED);
}
```

Figure 2 Modify check for protocol provider

Additionally, we need to determine if the callee is a SIP Bonjour contact, and if so, use the SIP stack we just created above instead of the default one

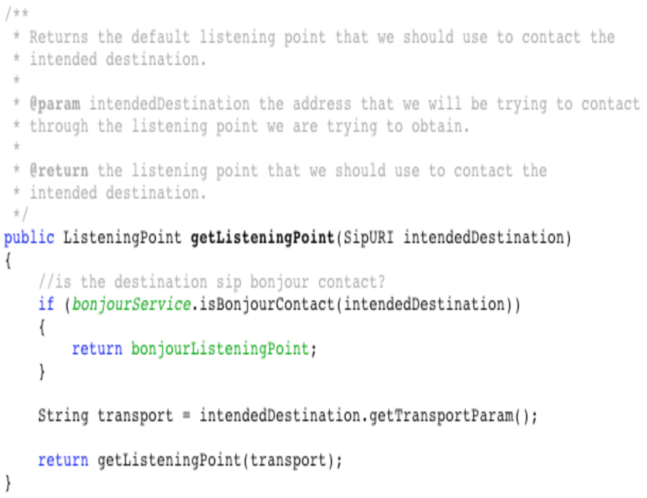


```
**
* Returns the provider appropriate for calling intended
* destination. For non SIP Bonjour destination it is
* equivalent to calling getJainSipProvider(getDefaultTransport())
*
* @return the Jain SipProvider appropriate for destination
*/
ublic SipProvider getDefaultJainSipProvider(Address destination)

   if (bonjourService.isBonjourContact(destination))
   {
       return bonjourSipProvider;
   }
   return getJainSipProvider(getDefaultTransport());
```

Figure 3 Modify getDefaultJainSipProvider()

A similar modification is required to the routine, which returns ListeningPoints.



```
/**
 * Returns the default listening point that we should use to contact the
 * intended destination.
 *
 * @param intendedDestination the address that we will be trying to contact
 * through the listening point we are trying to obtain.
 *
 * @return the listening point that we should use to contact the
 * intended destination.
 */
public ListeningPoint getListeningPoint(SipURI intendedDestination)
{
    //is the destination sip bonjour contact?
    if (bonjourService.isBonjourContact(intendedDestination))
    {
        return bonjourListeningPoint;
    }

    String transport = intendedDestination.getTransportParam();

    return getListeningPoint(transport);
}
```

Figure 4 Retrive ListeningPoint

3.4 Presence

Presence is out of scope of the I-D and therefore this implementation as well. However, since buddy list is involved in the implementation, a decision had to be made on how and when to show the contacts in the buddy list and what kind of presence information to give out as part of the Zeroconf advertisement.

It has to be kept in mind that the user has no control over who receives the service advertisement. It would be unwise to give out presence information to an unknown audience (specially because status messages can be personal in nature.)

Additionally, it is not known whether user is offline or there is no connectivity to the SIP proxy (which is a scenario covered by the I-D). As a result the SIP URI is advertised at all times the application is run. It is unclear what the presence status of the user is while the account if offline.

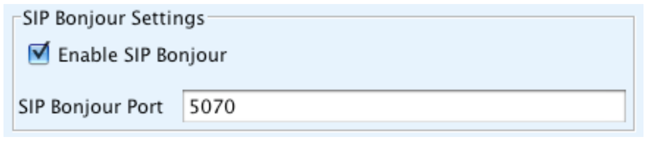
Due to these reasons, the SIP Bonjour contacts are available for calling, at all times the client software is running.

If the user does not want to give out service advertisement, he can always disable SIP Bonjour from the settings as explained in Section 4 or terminate the client.

4. CONFIGURATION

Zeroconf requires no configuration Very little configuration is required (and provided) in this implementation. The default settings would suffice most of the times.

There are two user configurable parameters. These can be manipulated at account creation time or modified at a later time. If changes are made to an existing account, the client must be restarted for changes to take effect. A screenshot of the configuration is shown below:



SIP Bonjour Settings

☑ Enable SIP Bonjour

SIP Bonjour Port: 5070

4.1.1 Enable SIP Bonjour

The user can disable SIP Bonjour if he so wishes. In this case, the SIP URI will not be advertised and associated JAIN SIP infrastructure will not be created.

4.1.2 SIP Bonjour Port

The default port for SIP Bonjour ListeningPoint is bound to port 5070. This can be changed depending on the users requirements. In fact, this change is required if the user wishes to run more that one SIP account at the same time. Since multiple ListeningPoints cannot be bound to the same port for a given protocol (UDP), changing the port number is a must in this scenario.

5. FUTURE DIRECTIONS

5.1 Code Enhancement

The default port that is selected for SIP Bonjour is 5070. This is relatively safe and chances of a conflict are minimal. On the IANA list, 5070 is listed as a well-known port for *VersaTrans Server Agent Service*. To remove chances of a conflict, the implementation could select a random unused port at startup.

5.2 Integration with Mainline

It is the author's intent to submit a patch file to the SIP Communicator team, for including this implementation as part of the standard software distribution.

6. ACKNOWLEDGMENTS

My sincere thanks to Prof. Henning Schulzrinne (hgs@cs.columbia.edu) and Jae Woo Lee (jae@cs.columbia.edu) for help during the development of this project.

7. REFERENCES

[1] Lee, J., et al., (2008) SIP URI Service Discovery using DNS-SD, Internet-Draft, [Online], Available: http://tools.ietf.org/html/draft-lee-sip-dns-sd-uri

[2] SIP Communicator, [Online], Available: http://sip-communicator.org

[3] Cheshire & Krochmal, (2008) mDNS, Internet-Draft, [Online], Available: http://files.multicastdns.org/draft-cheshire-dnsext-multicastdns.txt

[4] FSF High Priority Projects, [Online] Available: http://www.fsf.org/campaigns/priority.html

[5] Apache Felix, [Online], Available: http://felix.apache.org/site/index.html

[6] OSGi, [Online], Available: http://www.osgi.org/Main/HomePage

[7] JmDNS, [Online], Available: http://jmdns.sourceforge.net

[8] JAIN SIP, [Online], Available: https://jain-sip.dev.java.net