Karoo Bridge Cookbook

Introduction

Karoo Bridge is a SIP Session Border Controller developed by OSS Software Solutions. Unlike other similar commercial offerings, Karoo Bridge tries to reconcile the evils of B2BUA and the SIP standards. Thus, Karoo Bridge is much closer to being a proxy rather than being a switch. Primarily, Karoo Bridge solves the issues brought about by NAT using well tested far-end NAT traversal techniques as well as bridging between multi-homed network deployments. It is also capable of acting as a switch that would connect and register to ITSP while doing protocol corrections for features that are otherwise unsupported by the service provider. Karoo Bridge also boasts of its scripting API which allows administrators to design their own routing logic. This document aims to course through the basics of the API all the way to advanced scripting.

Initial Setup (/etc/karoo.conf.d/sip.cfg)

1.1. Installing Karoo Bridge

First, enable EPEL in your yum repository by following the instructions at

http://fedoraproject.org/wiki/EPEL

Download the repository file from http://bridge.ossapp.com for the latest version of Karoo Bridge.

```
# cd /etc/yum.repos.d
# wget http://bridge.ossapp.com/karoo/karoo-1.5.0-centos.repo
# yum install oss_karoo
```

The above Linux commands should have setup the repository and installed the binary for Karoo Bridge. The next command should install Karoo Bridge as a Linux service:

```
# chkconfig --add karood
# chkconfig karood on
# /etc/init.d/karood start
```

1.2. Configure SIP Transport

By default, the installation process should have configured Karoo Bridge to listen on port 5060 and bound to the default NIC of your server. If you need to bind to a different port or you need to add new SIP listeners you must edit the configuration file at /etc/karoo.conf.d/sip.cfg. It uses a C-style syntax as its format. You will see the following configuration entries at the top of sip.cfg file.

```
interfaces = (
    {
        default = true;
        ip-address = "%KAROO_HOST_IP%";
        external-address = "%KAROO_HOST_IP%";
        tcp-enabled = false;
        sip-port = 5060;
    } );
```

default: If this value is set to true, Karoo Bridge will default to this transport when sending out SIP requests. You can override the default in Flexi-Route (The scripting API for Karoo Bridge) later on if you intend to use Karoo Bridge in a multi-homed setup. You must only choose a single interface as the default.
ip-address: This is the IP address where Karoo Bridge will bind to.
external-address: The external address that Karoo Bridge will use if it is installed behind a Firewall. Karoo Bridge will use this address for Via, Contact and SDP so that responses will be routed correctly through the Firewall.
tcp-enabled: If set to true, Karoo Bridge will listen for both TCP and UDP connections. Take note that TCP can be problematic in terms of NAT traversal and Karoo Bridge TCP implementation requires that user agents using TCP must support persistent TCP connections. Otherwise, we recommend that TCP is disabled to ensure better far-end NAT support.
sip-port: The port where Karoo Bridge will listen for SIP connections. If Karoo Bridge is behind Firewall, this port must be mapped in the Firewall rules.
To add new listeners, to the interface set, simply add a new braced group to the interfaces collection and separate it from the previous one using a comma.

```
    interfaces = (
    {
        default = true;
        ip-address = "192.168.1.10";
        external-address = "192.168.1.10";
        tcp-enabled = false;
        sip-port = 5060;
    },
    {
        ip-address = "192.168.1.20";
        external-address = "192.168.1.20";
        tcp-enabled = false;
        sip-port = 5060;
    } );
```

When changing anything in sip.cfg, Karoo Bridge needs to be restarted for the changes to take effect. The following command will restart Karoo Bridge

```
      # /etc/init.d/karoo-init restart
```

1.3. Configure Karoo Bridge Behind a Firewall

We have already discussed how to set the external-address of the Firewall for the SIP transports. Make sure that the internal sip-port(s) are mapped accordingly in the firewall rules for both TCP and UDP. On top of this, the ports used by Karoo Bridge to proxy RTP must also be mapped in your firewall rules.

```
        rtp-proxy-port-base = 30000;
        rtp-proxy-port-max = 60000;
```

Locate the above parameters in sip.cfg and change their values accordingly if you need a custom range. Take note that RTP uses UDP as the protocol so take note of this when creating the firewall rules to map the internal RTP port range.

The TCP transport spawns separate TCP sessions per connection. You must also define a range for TCP in your firewall rules. Of course, if you did not enable TCP in the SIP transport interfaces, then there is no need to map this range in your TCP firewall rules.

```
        sip-tcp-port-base = 10000;
        sip-tcp-port-max = 15000;
```

1.4. Enabling The Internal STUN Server

Karoo Bridge comes with an internal STUN server. For STUN to work, Karoo Bridge must be installed with a publicly reachable pair of interfaces. STUN will not work if Karoo Bridge is installed behind a firewall. To enable STUN, simply populate the two parameters with the respective IP address of eth0 and eth1.

```
        stun-primary-ip = "200.210.200.200";
        stun-secondary-ip = "200.210.200.210";
```

The default port for STUN is 3478. If you need to specify a different port, you may simply append to port to the IP address like this: `stun-primary-ip = "200.210.200.200:3333";`

1.5. Guard Your Network Against DoS Attacks

Karoo Bridge comes with a light-weight algorithm that monitors the rate of SIP traffic from each IP address that sends traffic to your network. Karoo manages this table via the *packet-rate-ratio* parameter.

```
        packet-rate-ratio = "50/100/3600";
```

It works by detecting the packet read rate per second as designated by the upper limit. If the value of packet rate is 50/100, the maximum packet rate before the SBC raises the alert level if a potential denial of service attack is 100 packets per second. When this happens, the transport layer checks if there is a particular IP that is sending more than its allowable rate of 50 packets per second. If the sender is violating the threshold, it will be banned for 1 hour which is the third parameter of 3600 seconds.

One may statically define a list of known IP addresses "OR" networks so that they get immunity against the packet-rate-ratio algorithm. This would normally contain the IP addresses of known traffic sources or destinations like the local PBX or trunk gateways. For Call Centers with predictive or progressive dialers in the network, it would be wise to also white-list those applications.

```
    packet-rate-white-list = (
        {source-ip = "192.168.1.10";},
        {source-network = "192.168.1.1/24";}
    );
```

Note: At this point you would have sip.cfg fully configured. The rest of the parameters that are not discussed here must remain using their default value unless, otherwise, you think you have a good reason to do so. Mail support.ossapp.com if you have further questions.

Basic Flexi-Routing (/etc/karoo.conf.d/routes/route.js)

Karoo Bridge routing is done using an API called Flexi-Route. Flexi-Route gives you a handle on every SIP Message that goes in and out of Karoo Bridge. Flexi-Route is standard ECMA JavaScript so it benefits from every cool feature that the language provides including JavaScript regular expressions. The inspiration behind a scripting language versus using DB-based dial plans was born out of the fact that there is no single rule in routing and no amount of database structure would be enough to compensate for the different and ever changing rules needed for dial-plan maintenance.

2.1. Routing By Domain

To route SIP Messages based on domain is very easily done in Flexi-Route. First thing you need to do is to construct a JavaScript function that accepts one parameter and save it to a file with a .js extension. In our example we will save the js file as *mysipdomain.com.js*

File: mysipdomain.com.js

```
function is_mysipdomain_routable(profile)
{
}
```

The *profile* parameter holds the SIP Message and can be accessed using the dot notation *profile.sipMessage.*

There are many ways we can route SIP Messages to a particular SIP Server for a particular domain. We will start with the basic scenario that Karoo Bridge is configured to only listen on a single interface.

We will create a dial-plan rule that does the following

- Determine the IP Address of the sender.
- Determine the domain of the packet based on the from header
- If the packet has the correct domain and the IP address is not looping back to our SIP Server, retarget it towards our Server.

This can be written in Flexi-Route as follows:

File: mysipdomain.com.js

```
function is_mysipdomain_routable(profile)
{
      var ip = profile.sipMessage.getSourceAddress();
      var domain = profile.sipMessage.getFromHost();
      if (ip != “200.210.200.220” && domain == “mysipdomain.com”)
      {
            profile.setTargetAddress(
                  “udp”,
                  “pbx.mysipdomain.com”,
                  “5060”);
            return true;
      }else
      {
            return false;
      }
}
```

Save this file in /etc/karoo.conf.d/routes folder. At this point we now have a basic dial-plan for the domain “mysipdomain.com” based on the from host/domain of the SIP request.

The next thing to do is to enable this dial-plan in route.js. Open /etc/karoo.conf.d/routes/route.js using your favorite text editor. You should see a skeleton class like this:

File: /etc/karoo.conf.d/routes/route.js

```
function Route()
{
}

Route.prototype = new RouteProfile();

Route.prototype.isRoutable = function()
{
    return false;
}
```

To enable our newly created dial-plan, simply modify the prototype for the isRoutable() function as follows:

```
Route.prototype.isRoutable = function()
{
      if (is_mysipdomain_routable(this))
            return true;
      else
            return false;
}
```

2.2. Reloading Karoo Bridge After a Dial-plan Change

Unlike sip.cfg, changes to the Flexi-Route scripts do not require a restart. All that needs to be done is to let Karoo Bridge know that a change has occurred so it would reload the scripts. This can be done by issuing the command below:

```
# karoo --reload-config
```

To be sure there is no syntax error in your scripts, check the logs for JavaScript errors. The logs can be found in /var/log/karoo/sbc.log. You should see something similar to the log below if you have some typos in your script.

```
08:03:04.964:  Javascript error on line 9 : Route.prototype.routeRequest =
function()
08:03:04.967:     [CID=00000000] JS: ReferenceError: i_am_a_syntax_error is not
defined
{
undefined:9
Route.prototype.routeRequest = function()
     ^
ReferenceError: i_am_a_syntax_error is not defined
    at unknown source
}
```

Correct the syntax error in your script and reissue "`karoo --reload-config`".

2.3. Routing Using a Specific Numbering Scheme

Let us say your SIP Server is a PBX and you want to validate dial-strings using a numbering scheme currently being enforced by the PBX. Let us start with a 4 digit numbering scheme to make things simple. JavaScript has a rich set of string functions you could use to process dial-strings if it is present in a SIP Request. In this example we will be using some of these functions to check whether a dial-string has exactly 4 digits and starts with a 3 (3000-3999).

Note: Not all SIP Request has a dial-string. In particular REGISTER request will not have a user in the request-uri. So when using dial-strings as a filter for your dial-plans, you need to make sure that it only applies to INVITE requests.

Open /etc/karoo.conf.d/routes/mydomain.com.js again. We will be modifying the function to add some request-uri processing if the request is an INVITE. Replace the function with the code below. We will explain later what each new additional line means.

File: /etc/karoo.conf.d/routes/mysipdomain.com.js

```
function is_mysipdomain_routable(profile)
{
      var ip = profile.sipMessage.getSourceAddress();
      var domain = profile.sipMessage.getFromHost();
      var ds = profile.sipMessage.getRequestUriUser();
      var isInvite = profile.sipMessage.isRequest(“INVITE”);

      if (ip != “200.210.200.220” && domain == “mysipdomain.com”)
      {
            if (isInvite)
            {
                  if (typeof ds != “string”)
                        return false;

                  if (ds.length != 4)
                        return false;

                  if (ds.charAt(0) != '3')
                        return false;
            }

            profile.setTargetAddress(
                        “udp”,
                        “pbx.mysipdomain.com”,
                        “5060”);
            return true;
      }else
      {
            return false;
      }
}
```

You should have noticed that we merely added another “if” block within the outer “if” that checks for a valid domain. This dial-plan does an additional step in filtering request and can be defined in plain English as follows.

- Determine the IP Address of the sender.
- Determine the domain of the packet based on the from header
- Determine if the request is an INVITE and has dial-string falling under 3000-3999 range.
- If the packet has the correct domain and the IP address is not looping back to our SIP Server, retarget it towards our Server.

`if (isInvite)` - Checks if the request is an INVITE
`if (typeof ds != “string”)` - Checks whether the request-uri has a user
`if (ds.length != 4)` – Checks number of digits
`if (ds.charAt(0) != '3')` – Make sure it starts with 3

2.4. Modifying Target Domains

Let us assume that the sender of a SIP packet does not know the domain of our PBX. We therefore cannot determine the destination of the packet by domain but only through the numbering scheme. However, for our PBX to accept the call, we must send it the correct domain. We may reconstruct our function as follows:

File: /etc/karoo.conf.d/routes/mysipdomain.com.js

```
function is_mysipdomain_routable(profile)
{
      var ip = profile.sipMessage.getSourceAddress();
      var domain = profile.sipMessage.getFromHost();
      var ds = profile.sipMessage.getRequestUriUser();
      var isValidNumber = false;
      var isInvite = profile.sipMessage.isRequest(“INVITE”);

      if (  isInvite &&
            typeof ds == “string” &&
            ds.length == 4 &&
            ds.charAt(0) == '3')
      {
             isValidNumber = true;
      }

      if (ip != “200.210.200.220” && domain == “mysipdomain.com”)
      {
            if (isInvite && !isValidNumber)
                  return false;

            profile.setTargetAddress(
                        “udp”,
                        “pbx.mysipdomain.com”,
                        “5060”);
            return true;
      }else if (isValidNumber)
      {
            profile.setTargetDomain(“mysipdomain.com”);
            profile.setTargetAddress(
                        “udp”,
                        “pbx.mysipdomain.com”,
                        “5060”);
            return true;
      }
}
```

In the above example, we modified the previous script to allow INVITE requests with a valid number to be routed to our PBX for as long as it complies with the numbering scheme. Prior to routing we made sure we set the correct domain of the request by calling the function profile.setTargetDomain(). This will rewrite the domain of the request-uri, to-uri and from-uri respectively. Profile.setTargetDomain() is a convenience function. If you need to be specific about which uri to rewrite, then you can access the SIP Message directly. The if block above maybe rewritten as follows:

```
      }else if (isValidNumber)
      {
            var dm = “mysipdomain.com”
            profile.sipMessage.setRequestUriHostPort(dm);
            profile.sipMessage.setFromHostPort(dm);
            profile.sipMessage.setToHostPort(dm);
            profile.setTargetAddress(
                        “udp”,
                        “pbx.mysipdomain.com”,
                        “5060”);
            return true;
      }
```

2.5. Rewriting Dialed Numbers

Let us assume that we have a DID number 18004565000. We want to route this number to our default auto attendant at extension 3000. We can modify our rule as follows.

File: /etc/karoo.conf.d/routes/mysipdomain.com.js

```
      function is_mysipdomain_routable(profile)
      {
            var ip = profile.sipMessage.getSourceAddress();
            var domain = profile.sipMessage.getFromHost();
            var ds = profile.sipMessage.getRequestUriUser();
            var isValidNumber = false;
            var isInvite = profile.sipMessage.isRequest(“INVITE”);

            if (  isInvite &&
                  typeof ds == “string” &&
                  ds.length == 4 &&
                  ds.charAt(0) == '3')
            {
                   isValidNumber = true;
            }

            if (ip != “200.210.200.220” && domain == “mysipdomain.com”)
            {
                  if (isInvite && !isValidNumber)
                        return false;

                  profile.setTargetAddress(
                              “udp”,
                              “pbx.mysipdomain.com”,
                              “5060”);
                  return true;
            }else if (isValidNumber)
            {
                  profile.setTargetDomain(“mysipdomain.com”);
                  profile.setTargetAddress(
                              “udp”,
                              “pbx.mysipdomain.com”,
                              “5060”);
                  return true;
            }
            else if (ds == “1800456500”)
            {
                  profile.sipMessage.setRequestUriUser(“3000”);
                  profile.setTargetAddress(
                              “udp”,
                              “pbx.mysipdomain.com”,
                              “5060”);
                  return true;
            }
      }
```

We simply added a new else-if block almost similar to the valid number block but this time checking for a specific dialed number and set the request-uri-user to 3000. You may also rewrite the to-uri if needed by modifying the else-if block as follows:

```
            else if (ds == “1800456500”)
            {
                  profile.sipMessage.setRequestUriUser(“3000”);
                  profile.sipMessage.setToUser(“3000”);
                  profile.setTargetAddress(
                              “udp”,
                              “pbx.mysipdomain.com”,
                              “5060”);
                  return true;
            }
```

2.6. Debugging JavaScripts

Flexi-Route allows you to dump log information to /var/log/karoo/sbc.log. Three functions are exposed. log_info(), log_debug() and log_error(). log_debug() will only appear if the debug level of karoo is set to debug or trace priorities,

File: /etc/karoo.conf.d/routes/mysipdomain.com.js

```
      function is_mysipdomain_routable(profile)
      {
            var ip = profile.sipMessage.getSourceAddress();
            var domain = profile.sipMessage.getFromHost();
            var ds = profile.sipMessage.getRequestUriUser();
           var isValidNumber = false;
            var isInvite = profile.sipMessage.isRequest(“INVITE”);

            log_debug(“Received request from “ + ip + “ with domain “ << domain);

           if (   isInvite &&
                  typeof ds == “string” &&
                  ds.length == 4 &&
                  ds.charAt(0) == '3')
```

```
        {
                isValidNumber = true;
        }

    .
    .
    .
```

Advanced Flexi-Routing

So far we have learned how to create a dial-plan for a single PBX server using a single NIC. Flexi-Route is much better than that. In fact, if you are a bit of a JavaScript savvy, you must have some great routing scheme in mind after reading chapter 2.

3.1. Multiple Dial-plans

So far we have created a single JavaScript file (mysipdomain.com.js) and placed it to /etc/karoo.conf.d/routes folder. You can have as many of these scripts as needed for as long as your function name is unique for each dial-plan. Once you have created and added a new script to the route folder, all you need to do next is call your script from route.js. Below is an example of a multiple dial-plan

File: /etc/karoo.conf.d/routes/route.js

```
Route.prototype.isRoutable = function()
{
      if (is_pbx_call(this))
            return true;
      else if (is_idd_call(this)
            return true;
      else if (is_local_call(this)
            return true;
      else
            return false;
}
```

Each of these function call must be defined in their respective script and with their respective rules and filters.

3.2. Determining Interfaces Used

In chapter I, we have set the following multiple interfaces as our example.

```
interfaces = (
{
    default = true;
    ip-address = "192.168.1.10";
```

```
        external-address = "192.168.1.10";
        tcp-enabled = false;
        sip-port = 5060;
    },
    {
        ip-address = "192.168.1.20";
        external-address = "192.168.1.20";
        tcp-enabled = false;
        sip-port = 5060;
    } );
```

These interfaces will be exposed to Flexi-Route via the following array objects:

```
      var sip_interface_address = new Array();
      var sip_interface_port = new Array();
```

sip_interface_address: This array contains the internal interface addresses in exactly the same order as they were listed in sip.cfg

sip_interface_port: This array contains the SIP ports assigned to each interfaces in exactly the same order as they were listed in sip.cfg

To determine which listener was used by a particular SIP Message, we will be using the getInterfaceAddress() and getInterfacePort() member function of the SIP Message object as follows:

```
var ifaceAddr = profile.sipMessage.getInterfaceAddress();
var ifacePort = profile.sipMessage.getInterfacePort();
if (  ifaceAddr ==  sip_interface_address[0] &&
      ifacePort == sip_interface_port[0])
{
      // Do routing for messages received via the first interface
}
else if (   ifaceAddr == sip_interface_address[1] &&
            ifacePort == sip_interface_port[1])
{
      // Do routing for messages received via the second interface
}
```

3.3. Using Multiple Interfaces for Routing

Assume that we have 2 interfaces. The first one faces the WAN and the second one faces the local intranet. Our PBX is installed in the local intranet reachable via the second interface. With this in mind we know which packets came from the PBX and which packets are coming in from the WAN. In this example we will be introducing multiple functions to handle each of the dial-plan. Anything that has a valid domain and was received from the WAN goes to our PBX. Anything that is received from the LAN automatically goes to the trunk provider. Unknown domain received from the WAN which has a qualified number scheme is routed to our PBX. This can be represented in the script as follows:

File: /etc/karoo.conf.d/routes/mysipdomain.com.js

```
function is_mysipdomain_routable(profile)
{
      var ifaceAddr = profile.sipMessage.getInterfaceAddress();
      var ifacePort = profile.sipMessage.getInterfacePort();
      var ds = profile.sipMessage.getRequestUriUser();
      var domain = profile.sipMessage.getFromHost();
      if (is_my_pbx_routable(profile, ifaceAddr, ifacePort, ds, domain))
            return true;
      else if (is_my_sip_trunk_routable(profile, ifaceAddr, ifacePort, ds, domain))
            return true;

      return is_numbering_scheme_routable(
            profile, ifaceAddr, ifacePort, ds, domain);
}


function is_my_sip_trunk_routable(profile, ifaceAddr, ifacePort, ds, domain)
{
      if (typeof ds != “string”)
            return false;
      //
      // Check if the packet came in using the LAN interface
      //
      if (  ifaceAddr != sip_interface_address[1] ||
            ifacePort != sip_interface_port[1])
            return false;
      //
      // Route it to our ITSP
      //
      profile.setTargetDomain(“myitsp.com”);
      profile.setTargetAddress(“udp”, “gw.myitsp.com”, “5060”);
      //
      // Use the WAN interface to send it out
      //
      profile.setInterfaceAddress(sip_interface_address[0], sip_interface_port[0]);
      return true;
}

function is_my_pbx_routable(profile, ifaceAddr, ifacePort, ds, domain)
{
      if (typeof domain != “string”)
            return false;
      //
      // Check if the packet came in using the WAN interface
      //
      if (  ifaceAddr != sip_interface_address[0] ||
            ifacePort != sip_interface_port[0])
            return false;
      //
      // Check if the domain is valid
      //
      if (domain != “mysipdomain.com”)
            return false;
      //
      // route it to our PBX
      //
      profile.setTargetDomain(“mysipdomain.com”);
      profile.setTargetAddress(“udp”, “pbx.mysipdomain.com”, “5060”);
      //
      // Use the LAN interface to send it out
      //
      profile.setInterfaceAddress(sip_interface_address[0], sip_interface_port[0]);
      return true;
}
```

```
function is_numbering_scheme_routable(profile, ifaceAddr, ifacePort, ds)
{
      if (typeof ds != “string”)
            return false;
      //
      // This is a catch all handler. If it doesn't qualify here, the SIP Message
      // will get a 404
      //
      // Check if the packet came in using the WAN interface
      //
      if (  ifaceAddr != sip_interface_address[0] ||
            ifacePort != sip_interface_port[0])
            return false;
      //
      // Check if the numbering scheme is valid
      //
      if (ds.length != 4 && ds.charAt(0) != '3')
      {
            //
            // Check if it is the auto-attendant DID
            //
            if (ds == “1800456500”)
                  profile.sipMessage.setRequestUriUser(“3000”);
            else
                  return false;
      }
      //
      // route it to our PBX
      //
      profile.setTargetDomain(“mysipdomain.com”);
      profile.setTargetAddress(“udp”, “pbx.mysipdomain.com”, “5060”);
      //
      // Use the LAN interface to send it out
      //
      profile.setInterfaceAddress(sip_interface_address[0], sip_interface_port[0]);
      return true;
}
```

3.4. Authenticating Against a Trunk Provider Through Bridging

Our previous example assumes that the SIP Trunk provider will not authenticate our calls. This is called IP Auth in trunking lingo. However, if there is a need for you to authenticate against your provider, you can use the bridge() function to assign credentials to the SIP Request. The following function shows how this method is used.

```
function is_my_sip_trunk_routable(profile, ifaceAddr, ifacePort, ds, domain)
{
      if (typeof ds != “string”)
            return false;
      //
      // Check if the packet came in using the LAN interface
      //
      if (  ifaceAddr != sip_interface_address[1] ||
            ifacePort != sip_interface_port[1])
            return false;
      //
      // Route it to our ITSP
      //
      profile.setTargetDomain(“myitsp.com”);
      profile.setTargetAddress(“udp”, “gw.myitsp.com”, “5060”);
      //
      // Use the WAN interface to send it out
      //
      profile.setInterfaceAddress(sip_interface_address[0], sip_interface_port[0]);


      //
      // Assign credentials for authentication
      //
      profile.bridge(“myaccount”, “mypassword”);
      return true;
}
```

Note: The bridge function does more than assign authentication credentials. Bridging calls to an ITSP also gives the call the ability to shield the ITSP from mid-dialog advanced feature like call transfers. Most ITSP do not support REFER. Bridging the call will make Karoo Bridge handle the REFER locally so that it does not propagate to the ITSP which may not support it.

3.5. Sending REGISTER To an ITSP

Some ITSP requires you to send a REGISTER to them before you can access their service. Karoo Bridge has a facility to do this via Flexi-Route. Open /etc/karoo.conf.d/routes/route.js using your favorite text editor and modify it as follows:

File: /etc/karoo.conf.d/routes/route.js

```
      function Route()
      {
            this.sendRegister("myuser", "myitsp.com", "mypass", "proxy.myitsp.com", 1800, 0);
      }

      Route.prototype = new RouteProfile();

      Route.prototype.isRoutable = function()
      {
            if (is_mysipdomain_routable(this))
                  return true;
            else
                  return false;
      }
```

The sendRegister() function is defined as follows

```
      sendRegister (user, domain, password, proxy, expires, interfaceIndex)
```

user: The account to be used
domain: The domain or IP address of the ITSP
password: Password to be used for the account
proxy: The proxy of the ITSP.
expires: Expiration value expressed in seconds
interfaceIndex: The listener index to be used

3.6. Using Dynamic Conferences

Most PBX already have conferences. However, sometimes we want the ability to hold meetings on the fly using temporary conference rooms that come and go as members connect and disconnect to the bridge. It is very easy to do this in Karoo Bridge. We will assume that our dynamic conference number scheme is a 5 digit number all starting with the prefix 55. We will be introducing a new function call in mysipdomain.js named is_my_conference_routable:

```
function is_my_conference_routable(profile, ifaceAddr, ifacePort, ds)
{
      //
      // Check if a dial-string is present
      //
      if (typeof ds != “string”)
            return false;
      //
      // Check if the request is an INVITE
      //
      if (!profile.sipMessage.isRequest(“INVITE”))
            return false;
      //
      // Check if the packet came in using the LAN interface
      //
      if (  ifaceAddr != sip_interface_address[1] ||
            ifacePort != sip_interface_port[1])
            return false;
      //
      // Check if the domain is valid
      //
      if (domain != “mysipdomain.com”)
            return false;
      //
      // Check the numbering scheme is valid
      //
      if (ds.length != 5 || ds.substr(0, 2) != “55”)
            return false;
      //
      // Everything is set.  Bridge it to the dynamic conference.
      //
      profile.bridgeToConference(ds);
      return true;
}
```

You may also assign a PIN to the conference for an extra level of security. Simply call the bridgeToConference() as follows:

```
      profile.bridgeToConference(ds, “4567”);
```

In this function, any one who dials 55000 – 55999 from the PBX will be routed to a dynamic conference.

3.7. Bridging It All Together – sipXecs Recipe

In this chapter, we will apply what we have learned to create dial-plans that will work with sipXecs. Although sipXecs has a built-in SBC and far-end NAT traversal capability, it lacks some of the functionalities that Karoo Bridge can offer like support for muti-homing and scriptability.

The steps to use Karoo Bridge as an SBC for sipXecs is quite simple:

- Create a phantom user in sipXecs to be used by Karoo Bridge to authenticate itself with the system.
- Create an unmanaged gateway in sipXecs and point it to the LAN address of Karoo Bridge
- Create custom dial-plans for international dialing and point it the the unmanaged gateway assigned to Karoo.

We will assume that all SIP Trunks will be defined in Flexi-Route. Thus, it is no longer necessary for you to define the dial-plans in sipXecs for your trunking requirements. All routing would be done in Karoo Bridge. Karoo bridge will also be responsible for remote workers. Thus it is no longer necessary for sipXecs to serve as a media relay.

Setup Information

Karoo Bridge
Interface 1: 192.168.1.10 (WAN)
Interface 2: 10.0.0.10 (LAN)
Firewall: 200.100.200.100
Domain: mysipdomain.com (external DNS)
Host: sbc.mysipdomain.com
ITSP-Account: 7773456:mypass
ITSP-Gateway: 100.200.100.200
ITSP-Proxy: 100.200.100.200

sipXecs
Interface 1: 10.0.0.20
Domain: mysipdomain.com (internal DNS)
Host: pbx.mysipdomain.com
Numbering: 3000-3999
Auto-Attendant: 3000

sipXecs Karoo Account

user: 3030
password: karoo

Prepare Karoo Bridge Transport

Follow the steps in chapter 1 in configuring sip.cfg. Set the interface array as follows.

```
interfaces = (
{
    default = true;
    ip-address = "192.168.1.10";
    external-address = "200.100.200.100";
    tcp-enabled = false;
    sip-port = 5060;
},
{
    ip-address = "10.0.0.10";
    tcp-enabled = false;
    sip-port = 5060;
} );
```

Prepare Karoo Bridge route.js

Open /etc/karoo.conf.d/routes/route.js and modify accordingly:

File: /etc/karoo.conf.d/routes/route.js

```
function Route()
{
      this.sendRegister(
            "7773456", "100.200.100.200", "mypass", "100.200.100.200", 1800, 0);
}

Route.prototype = new RouteProfile();

Route.prototype.isRoutable = function()
{
      if (is_mysipdomain_routable(this))
            return true;
      else
            return false;
}
```

Prepare The Flexi-Route Dial-plans

Create script file in /etc/karoo.conf.d/routes/mysipdomain.com.js and populate it with the following code:

File: /etc/karoo.conf.d/routes/mysipdomain.js

```
function is_mysipdomain_routable(profile)
{
      var ifaceAddr = profile.sipMessage.getInterfaceAddress();
      var ifacePort = profile.sipMessage.getInterfacePort();
      var ds = profile.sipMessage.getRequestUriUser();
      var domain = profile.sipMessage.getFromHost();

      if (is_my_pbx_routable(profile, ifaceAddr, ifacePort, ds, domain))
            return true;
      else if (is_my_sip_trunk_routable(profile, ifaceAddr, ifacePort, ds, domain))
            return true;
      else if (is_my_conference_routable(
            profile, ifaceAddr, ifacePort, ds, domain))

      return is_numbering_scheme_routable(
            profile, ifaceAddr, ifacePort, ds, domain);
}


function is_my_sip_trunk_routable(profile, ifaceAddr, ifacePort, ds, domain)
{
      if (typeof ds != “string”)
            return false;
      //
      // Check if the packet came in using the LAN interface
      //
      if (  ifaceAddr != sip_interface_address[1] ||
            ifacePort != sip_interface_port[1])
            return false;
      //
      // Route it to our ITSP
      //
      profile.setTargetDomain(“100.200.100.200”);
      profile.setTargetAddress(“udp”, “100.200.100.200”, “5060”);
      //
      // Use the WAN interface to send it out
      //
      profile.setInterfaceAddress(sip_interface_address[0], sip_interface_port[0]);
      //
      // Bridge it using our account
      //
      profile.bridge(“7773456”, “mypass”);
      return true;
}

function is_my_pbx_routable(profile, ifaceAddr, ifacePort, ds, domain)
{
      if (typeof domain != “string”)
            return false;
      //
      // Check if the packet came in using the WAN interface
      //
      if (  ifaceAddr != sip_interface_address[0] ||
            ifacePort != sip_interface_port[0])
            return false;
      //
      // Check if the domain is valid
      //
      if (domain != “mysipdomain.com”)
            return false;
      //
      // route it to our PBX
      //
      profile.setTargetDomain(“mysipdomain.com”);
      profile.setTargetAddress(“udp”, “10.0.0.20”, “5060”);
      //
      // Use the LAN interface to send it out
      //
      profile.setInterfaceAddress(sip_interface_address[0], sip_interface_port[0]);
      //
      // Check if the source address is the ITSP.  If yes, bridge it
      //
      if (profile.sipMessage.getSourceAddress() == “100.200.100.200”)
            profile.bridge(“3030”, “karoo”);
      return true;
}

function is_numbering_scheme_routable(profile, ifaceAddr, ifacePort, ds, domain)
{
      if (typeof ds != “string”)
            return false;
      //
      // This is a catch all handler. If it doesn't qualify here, the SIP Message
      // will get a 404
      //
      // Check if the packet came in using the WAN interface
      //
      if (  ifaceAddr != sip_interface_address[0] ||
            ifacePort != sip_interface_port[0])
            return false;
      //
      // Check if the numbering scheme is valid
      //
      if (ds.length != 4 && ds.charAt(0) != '3')
      {
            //
            // Check if it is the auto-attendant DID
            //
            if (ds == “1800456500”)
                  profile.sipMessage.setRequestUriUser(“3000”);
            else
                  return false;
      }
      //
      // route it to our PBX
      //
      profile.setTargetDomain(“mysipdomain.com”);
      profile.setTargetAddress(“udp”, “10.0.0.20”, “5060”);
      //
      // Use the LAN interface to send it out
      //
      profile.setInterfaceAddress(sip_interface_address[0], sip_interface_port[0]);
      //
      // Check if the source address is the ITSP.  If yes, bridge it
      //
      if (profile.sipMessage.getSourceAddress() == “100.200.100.200”)
            profile.bridge(“3030”, “karoo”);
      return true;
}

function is_my_conference_routable(profile, ifaceAddr, ifacePort, ds, domain)
{
      //
      // Check if a dial-string is present
      //
      if (typeof ds != “string”)
            return false;
      //
      // Check if the request is an INVITE
      //
      if (!profile.sipMessage.isRequest(“INVITE”))
            return false;
      //
      // Check if the packet came in using the LAN interface
      //
      if (  ifaceAddr != sip_interface_address[1] ||
            ifacePort != sip_interface_port[1])
            return false;
      //
      // Check if the domain is valid
      //
      if (domain != “mysipdomain.com”)
            return false;
      //
      // Check the numbering scheme is valid
      //
      if (ds.length != 5 || ds.substr(0, 2) != “55”)
            return false;
      //
      // Everything is set.  Bridge it to the dynamic conference.
      //
      profile.bridgeToConference(ds);
      return true;
}
```