

La Guía OpenR2 – Enero 2009 - Versión 0.1

Moisés Silva <moy@sangoma.com>

Alexandre Alencar <alexandre.alencar@gmail.com>

Traducción

Juan Carlos Huerta <juan.huerta@nucleum.com.mx>

1. Acerca de esta guía.

OpenR2 es una librería que implementa señalización MFC/R2 sobre líneas E1 usando la interfaz de telefonía Zapata, DAHDI y próximamente la librería de abstracción TDM OpenZAP. El presente documento NO describe las interfaces de programación provistas por OpenR2, para esto hay que referirse al código :-), y no dudes en preguntarme cualquier pregunta que tengas.

Éste documento describe la instalación de la librería usada por Asterisk y/o FreeSwitch.

Para preguntas relacionadas a temas de instalación o implementación, por favor no me contactes directamente, a menos que estés dispuesto a pagarme por soporte :-). En lugar, usa la lista de correo asterisk-r2 de Digium en: <http://lists.digium.com/mailman/listinfo/asterisk-r2>, o únete al canal #openr2 de IRC en el servidor irc.freenode.org. Normalmente me encontrarás bajo el nick de 'moy'.

Si hay alguna variante de MFC/R2 que no se encuentre soportada que necesitas (o no funciona como debería), mándame un correo electrónico. Apreciaré también cualquier contribución o corrección a éste documento o al mismo código.

2. ¿Qué es MFC/R2?

MFC/R2 es una señalización telefónica usada ampliamente en México, Brasil y otros países de Latinoamérica y Asia. Las iniciales de MFC/R2 provienen de Multi Frequency Compelled R2 (R2 dirigido por multifrecuencia). Comparado con protocolos de señalización más recientes como ISDN PRI/BRI o SS7, R2 ofrece funcionalidades limitadas. La señalización es sólo usada para establecer la llamada o para terminarla. Algunas de las variantes MFC/R2 envían pulsos de cobranza durante la llamada, pero es raramente usado.

Existen versiones analógicas y digitales de MFC/R2, cualquier referencia a MFC/R2 o R2 en éste documento refieren a la versión digital que usan instalaciones E1. La mayoría de los lectores no tendrán interés en la versión analógica por su poco uso en la actualidad. MFC/R2 es definido por la ITU (International Telecommunication Unit), sin embargo la mayoría de los países que usan MFC/R2 no siguen las especificaciones de la ITU e implementan su propia variante.

3. ¿Cómo funciona MFC/R2?

Sin importar si eres un programador que va a usar la librería o un administrador de sistemas que va a instalarla con una plataforma de telefonía (ej: Asterisk o FreeSwitch), es buena idea tener el conocimiento básico de cómo funcionan las señales, que te permitirán resolver problemas con facilidad. Si no estas interesado en ésto y sólo quieres una guía rápida para instalar OpenR2 y dejarla funcionando con la plataforma de telefonía de tu elección, puedes saltarte ésta sección.

MFC/R2 es un protocolo de señalización peer-to-peer, que significa que solo hay dos participantes involucrados en un enlace E1 R2 y ambas puntas se comportan del mismo modo, al contrario de PRI donde se tiene la parte “NET” y la parte “CPE” del enlace.

Como se mencionó, MFC/R2 significa Multi Frequency Compelled R2. El nombre describe la naturaleza de ésta señalización, donde se tienen dos tipos de señales:

Señales de Línea. Son usadas para monitorear el estado de la llamada y las señales MF son usadas para transmitir información de la llamada durante el establecimiento de la misma (DNIS, ANI, Calling Party Category). Las señales de línea se envían usando señales CAS que viajan usando el canal 16 del enlace E1. Todas las señales CAS para cada canale del E1 es multiplexado por éste canal. Cada 2 milisegundos cada punta del enlace actualiza sus 4 bits de señal CAS conocidos como los bits ABCD.

MFC/R2 use 2 de esos 4 bits para enviar las siguientes señales: *Idle, Block, Seize, Seize Ack, Clear Back, Forced Release, Clear Forward, Answer*. Ya que sólo 2 bits se usan para 7 posibles señales es imposible no repetir algún patrón, es por eso que algunas de las 7 señales tienen el mismo patrón de bits, pero no representa un problema considerando que, por ejemplo, no puedes ir del estado *Idle* al *Forced Release*, por lo tanto, aunque el patrón de bits para *Forced Release* y *Seize* son los mismos, el protocolo conoce lo que la otra punta del enlace quiere. La razón de usar sólo 2 bits teniendo 4 disponibles es histórica y proviene de la época donde la versión analógica de MFC/R2 fue portada para trabajar en un mundo digital.

La siguiente tabla describe los patrones de bits usados en la señalización R2 mediante los bits ABCD de CAS.

Circuit State	Forward AB	Backward AB
Idle/Released	1 0	1 0
Seized	0 0	1 0
Seizure Acknowledged	0 0	1 1
Answered	0 0	0 1
Clear Back	0 0	1 1
Clear Forward (before Clear Back)	1 0	0 1
Clear Forward (after Clear Back)	1 0	1 1

Blocked	1 0	1 1
---------	-----	-----

Las **señales de direccionamiento**, por el otro lado, son 15 diferentes señales MF, que son tonos audibles compuestos por 2 frecuencias que viajan usando el canal de audio, y es por eso que un detector de audio es un componente importante en el paquete R2, OpenR2 por omisión usa su propio detector MF de R2, tomado prestado de la librería SpanDSP de Steve Underwood. No es necesario instalar SpanDSP para usar OpenR2, ya que el detector se encuentra “interconstruido”.

La ITU define qué frecuencias pueden ser mezcladas para componer los tonos MF y asigna los significados de cada uno de ellos. Sin embargo, algunos países asignan diferentes significados a estos tonos MF. Los tonos MF son identificados en la librería OpenR2 usando los números del 1 al 0 (0 siendo 10) y letras de la B a la F (la A no es usada, en su lugar se usa 0). Si habilitas las trazas en el módulo OpenR2 verás los detalles de qué tonos son enviados y recibidos durante cada establecimiento de llamada. Más información al respecto en la sección de “Soporte”, o en la sección de instalación de tu plataforma de telefonía de tu elección.

Los tonos de multifrecuencia (MF) son usados para transmitir ANI (Automatic Number Identification, comúnmente conocido como Identificador de Llamadas), DNIS (Dialed Number Identification Service, que es, el número marcado o número destino) y la Categoría de Marcado. Probablemente otra información es intercambiada en enlaces internacionales R2 (sin embargo no tengo experiencia con dichos enlaces y probablemente ya no se usan). Tan pronto como la llamada es aceptada o rechazada, el detector MF es apagado y no se intercambiarán más señales MF, es entonces que el canal de audio puede ser usada para voz.

4. Instalación de Zaptel/DAHDI.

La instalación de OpenR2 es muy sencilla, sin embargo hay algunos pre-requisitos que debes cumplir. Hasta ahora, la librería requiere Zaptel 1.2/1.4 o DAHDI para compilar. El script de configuración detectará si tienes instalado Zaptel 1.2, Zaptel 1.4 o DAHDI. Entrarás en problemas si tienes más de uno de esos paquetes instalados, ya que OpenR2 compilará con el primero que encuentre.

Zaptel 1.2 instala el encabezado de zaptel.h en /usr/include/linux/

Zaptel 1.4 instala el encabezado de zaptel.h en /usr/include/zaptel/

DAHDI instala el encabezado de user.h en /usr/include/dahdi/

Las rutas de las carpetas de instalación pueden variar, es buena idea asegurarse que **sólo** tienes **una** instalación. Si no tienes Zaptel o DAHDI instalado, ve a <http://downloads.digium.com/> para obtenerlo.

Si tienes planeado instalar Asterisk 1.2, necesitarás Zaptel 1.2

Si tienes planeado instalar Asterisk 1.4, necesitarás Zaptel 1.4 or DAHDI.

Si tienes planeado instalar Asterisk 1.6, necesitarás DAHDI.

== **Zaptel 1.2** ==

```
# wget http://downloads.digium.com/<ruta-a-zaptel-1.2>.tar.gz
# tar -xvpzf zaptel-<versión>.tar.gz
# cd zaptel-<versión>
# make
# make install
```

== **Zaptel 1.4** ==

```
# wget http://downloads.digium.com/<ruta-a-zaptel-1.4>.tar.gz
# tar -xvpzf zaptel-<versión>.tar.gz
# cd zaptel-<versión>
# make
# make install
```

== **DAHDI** ==

A partir del lanzamiento de DAHDI, Digium separó las herramientas del espacio de usuario y los controladores de kernel en 2 paquetes: dahdi-tools and dahdi-linux.

Primero debes instalar dahdi-linux, que contiene los controladores actuales, después dahdi-tools.

```
# wget http://downloads.digium.com/<ruta-a-dahdi>.tar.gz
# tar -xvpzf dahdi-linux-<versión>.tar.gz
# cd dahdi-linux-<versión>
# make
# make install
```

Ahora deberás descargar e instalar dahdi-tools. OepnR2 no requiere estrictamente dahdi-tools, sin embargo dahdi-tools provee varios binarios útiles para resolver problemas (como el reemplazo de zttool llamado dahdi-tool).

```
# wget http://downloads.digium.com/<ruta-a-dahdi>.tar.gz
# tar -xvpzf dahdi-tools-<versión>.tar.gz
# cd dahdi-tools-<versión>
# ./configure --prefix=/usr
# make
# make install
```

Ahora, si tienes una tarjeta Sangoma, necesitarás instalar wanpipe. Los controladores de sangoma se agregan automáticamente a Zaptel/DAHDI, pero necesitamos instalarlos primero. Los controladores de Sangoma se encuentran en un paquete llamado wanpipe. Consulta la wiki de Sangoma (<http://wiki.sangoma.com/>) para más información.

Ahora que ya tienes instalado Zaptel o DAHDI, necesitas configurar tu hardware. DAHDI es configurado en `/etc/dahdi/system.conf` y Zaptel es configurado en `/etc/zaptel.conf`. Afortunadamente para nosotros, `/etc/dahdi/system.conf` soporta sintaxis de la versión anterior (además de nuevas opciones). Recuerda que en el mundo R2, el enlace físico es E1, no T1 (al menos no he visto instalaciones R2 usando algo diferente a E1), si recibiste tus tarjetas desde Estados Unidos o Canadá, probablemente te las manden configuradas para T1. Las tarjetas Sangoma pueden cambiar esto por software, sin embargo las tarjetas Digium necesitan cambiar la opción de T1/E1 mediante jumpers en la tarjeta. La mejor forma de determinar si el span está configurado en modo E1 o T1 es leyendo el contenido del archivo `/proc/zaptel/<número de span>` o `/proc/dahdi/<número de span>` mediante el comando 'cat', tal como esto:

```
# cat /proc/zaptel/1
```

si tienes DAHDI.

```
# cat /proc/dahdi/1
```

Esto enlistará la configuración del span 1, para un E1 que enlistará 31 canales.

Un span E1 DAHDI o Zaptel se configura de la siguiente manera:

```
span=1,1,0,cas,hdb3
```

```
cas=1-15:1101
```

```
dchan=16
```

```
cas=17-31:1101
```

Es importante que conozcas el significado de cada parámetro. Describiré brevemente cada uno, pero te recomiendo que leas `zaptel.conf.sample` o `system.conf.sample` de tu distribución Zaptel/DAHDI.

Las líneas “span” determinan, por supuesto, la configuración del span. Puedes visualizar un “span” como un grupo de canales. Puedes ver cuántos spans se encuentran registrados enlistando el contenido del directorio `/proc/zaptel` o `/proc/dahdi`. Cada archivo enlistado en el directorio representa un span, El que el span represente un grupo físico de canales o una emulación por software de canales depende en el tipo de span. Los módulos de Kernel como `ztdummy` registran un span “simulado” sólo para proveer un reloj de sincronización (usando los recursos de tiempo del Kernel). Ok, suficiente acerca de spans, sólo recuerda que si quieres configurar alguno de los spans enlistados en `/proc/zaptel` o `/proc/dahdi` necesitarás modificar `/etc/zaptel.conf` o `/etc/dahdi/system.conf` usando el parámetro span. El parámetro span se compone de los siguientes elementos:

```
span=<número de span>,<fuente de sincronización>,<line build out (LBO)>,<framing>,<codificación>[,crc4|yellow[, yellow]]
```

Para propósitos de líneas R2, siéntete libre de ignorar cualquier cosa después del valor <codificación>. Describamos los demás:

<número de span> refiere a un número entero que representa el número de span. En nuestro ejemplo configuramos el span 1.

<fuente de sincronización> refiere a un entero que determina si la punta remota (independientemente de lo que tengas conectado del otro lado) proveerá la señal de reloj y será usada como fuente maestra para la sincronización del reloj. Si estas conectado a la red pública (PSTN, tu proveedor) normalmente necesitarás establecer el valor a un número mayor a 0, dependiendo en cuántos spans tienes. Pro ejemplo, si tienes 2 spans conectados a la PSTN, uno de los spans deberá estar configurado con <fuente de sincronización> en 1, y el siguiente en 2, es decir, primera fuente de sincronización y segunda fuente de sincronización respectivamente. Si tienes el span conectado a algún PBX viejo, probablemente el PBX viejo espere recibir el reloj de la PSTN, pero ya que no está conectado a la PSTN, sino a tu caja, deberás proveer el reloj a dicho PBX, en este caso configura éste valor a 0.

span=1,0,0,cas,hdb3

Esto significa que el span 1 proveerá el reloj a lo que encuentre conectado en la otra punta. Ten en cuenta que configurar incorrectamente éste valor puede causar ruidos/clics en el audio, faxes de pobre calidad o fallidos, operación desconfiable de modems entre otros problemas.

<line build out (LBO)> es un entero entre 0 y 7 que determina el nivel de transmisión del span, puedes escoger el valor dependiendo de la tabla provista en el archivo ejemplo de configuración de Zaptel/DAHDI:

0: 0 db (CSU) / 0-133 feet (DSX-1)
1: 133-266 feet (DSX-1)
2: 266-399 feet (DSX-1)
3: 399-533 feet (DSX-1)
4: 533-655 feet (DSX-1)
5: -7.5db (CSU)
6: -15db (CSU)
7: -22.5db (CSU)

<framing> y <codificación> determinan la señalización de nivel bajo y el formato de las tramas E1, para el caso de la señalización R2 usaremos cas y hdb3 respectivamente.

Eso fue para el “span”, ahora, vamos a ver que significa la línea “cas”. Después de configurar el span, necesitamos configurar cada canal del span con la señalización apropiada. En este caso, el span tendrá todos los canales con señalización cas (conocida también como señalización de usuario).

cas=x-y:1101

Significa que el rango de canales especificado de x a y reportará bits CAS. El :1101 es la posición inicial de los bits CAS (conocidos también como los bits ABCD) que son usados por R2 para señalar el estado de la línea (contestada, colgada, etc.). La señal 1101 significa “bloqueado” en R2, que significa que cuando el hardware despierta por primera vez no podrá recibir llamadas (Asterisk o cualquier otro software que use R2 después cambiarán los bits a 1001, que significan IDLE, o listo para hacer llamadas).

El parámetro dchan=16 ha sido un poco controversial desde que algunas personas no lo establecen cuando otros sí. El canal 16 (mejor dicho, timeslot 16) es usado para transmitir los bits CAS multiplexados para canales E1 restantes. En mi experiencia los bits son bien recibidos estableciendo o no dchan=16, pero establecerlo ayuda a limpiar cualquier señal previa en el canal que pueda haber estado. Yo recomiendo establecerlo a menos que tengas problemas al ejecutar “ztcfg” o “dahdi_cfg”. Algunas versiones de ztcfg no aceptarán un span con dchan=16 y rechazarán la configuración de los canales, sólo quita el parámetro para corregirlo.

4.1 Instalación Sangoma Wanpipe.

Este paso es solo requerido si tienes hardware Sangoma. Ya que el hardware de Sangoma, por definición, no es una tarjeta de telefonía basada de Zapata, requiere algunos pasos de instalación extra para integrar completamente las tarjetas Sangoma con la infraestructura de kernel de Zaptel.

Para mayor información acerca de la instalación de Wanpipe consulta la siguiente liga:

<http://wiki.sangoma.com/wanpipe-linux-asterisk-install>

5. OpenR2.

Ahora que tenemos la configuración del hardware en su lugar, podemos proceder a descargar e instalar OpenR2. El sitio oficial de OpenR2 es <http://www.libopenr2.org/>, sin embargo, el código fuente se encuentra en el sitio de código de Google en <http://code.google.com/p/openr2/> y la sección de descargas es <http://code.google.com/p/openr2/downloads/list>

1. Descargar OpenR2 de la página de Google Code.

```
# wget http://openr2.googlecode.com/files/openr2-1.0.0.tar.gz  
# tar -xzf openr2-1.0.0.tar.gz  
# cd openr2
```

2. Ejecutar el script de configuración.

```
# ./configure --prefix=/usr
```

La opción `--prefix=/usr` instalará la librería en `/usr/lib/`, pero si no especificas una opción `--prefix` se instalará en `/usr/local/`, yo recomiendo instalarlo usando `--prefix=/usr`, créeme, te ahorrará algunos problemas (la mayoría del tiempo). Si el script de configuración no encuentra

las cabeceras de Zaptel o DAHDI marcará un error. Asegúrate de buscar errores de configuración e instalar cualquier dependencia que te pueda hacer falta. En general, sólo Zaptel o DAHDI es requerido.

3. Compilar.

```
# make
```

Si no se marcan errores en este paso, continua. Cualquier error en esta etapa la mayoría de las veces significa un bug o que estas compilando en una plataforma no soportada. Puedes pedir por soporte en las listas o en IRC (ver 1. Acerca de ésta guía). Ten en cuenta que deberás proporcionar la descripción del error en la salida de la compilación y proveerlo para obtener soporte.

4. Intalar.

```
# make install
```

Eso es, ya estas listo para seguir. Recuerda que usualmente necesitarás ser root para ejecutar éste último paso, a menos que especifiques dentro de la opción “--prefix” una carpeta que tenga los privilegios de escritura para tu usuario.

Ahora que la librería se encuentra instalada, puedes proceder a instalar Asterisk. Sin embargo, puedes probar tus líneas R2 sin que Asterisk se encuentre involucrado. Lee la sección 6 “OpenR2 - Pruebas” para aprender como probar sin Asterisk. Si no estas interesado en correr pruebas, puedes proceder a la sección 7 “OpenR2 en Asterisk”.

6. OpenR2 - Pruebas

La librería OpenR2 provee un pequeño programa llamado “r2test” (a menos que hayas especificado la opción `--without-r2test`). Es una pequeña aplicación que usa un archivo de configuración para “escuchar” llamadas en los dispositivos Zaptel/DAHDI que especifiques, así como contestar llamadas y poder ponerlas en una aplicación tipo “echo” o reproducir archivos en formato alaw. El programa r2test no solo espera por llamadas, también puede ser configurado para realizar llamadas.

Para poder usar r2test necesitarás primero crear el archivo de configuración. Un ejemplo del archivo de configuración es incluido dentro de OpenR2 en la ruta `doc/r2test.conf`. El formato del archivo es:

```
# esto es un comentario
parámetro1=valor
parámetro2=valor
channel=inicio-fin
```

Cuando r2test encuentra un parámetro del tipo “channel”, creará un nuevo bloque de canales Zaptel/DAHDI, empezando con “inicio” y terminando con “fin” y los configurará con los parámetros precedentes. Debes estar seguro que “inicio” y “fin” sean dispositivos válidos Zaptel/DAHDI. Por ejemplo, ésta es una configuración mínima para r2test:

```
variant=mx  
caller=no  
maxani=10  
maxdnis=4  
channel=1-15
```

Un archivo r2test.conf que dicho contenido abrirá los canales Zaptel/DAHDI del 1 al 15, y los configurará con caller=no, maxani=10 y maxdnis=4. Si obtienes errores del tipo “No such file or directory”, puede ser por que no tienes cargados los controladores de Zaptel/DAHDI. Cuando los controladores se encuentran cargados, se generan dispositivos en /dev/zap o /dev/dahti. En el ejemplo de configuración previo los dispositivos /dev/zap/1 al /dev/zap/15 o /dev/dahti/1 al /dev/dahti/15 son requeridos.

El parámetro “variant” determina que variante R2 seá usada. Puedes enlistar las variantes soportadas con el comando:

```
# r2test -l
```

Par OpenR2 versión 1.0.0 la lista es:

```
Variant Code Country  
AR Argentina  
BR Brazil  
CN China  
CZ Czech Republic  
CO Colombia  
EC Ecuador  
ITU International Telecommunication Union  
MX Mexico  
PH Philippines  
VE Venezuela
```

El parámetro “caller” determina si el grupo de canales recibirán o realizarán llamadas. “caller=yes” significa que los canales generarán llamadas, en ese caso el parámetro “dnid” deberá ser especificado.

Los parámetros “maxani” y “maxdnis” sólo determinan el máximo de número de dígitos que serán recibidos para ANI y DNIS.

El parámetro “channel” ***deberá*** estar en el formato “x-y” para especificar el rango de canales a los cuales se les aplicarán las configuraciones previas (tal como el archivo zapata.conf o chan_dahdi.conf). Si sólo necesitas 1 canal deberás especificar algo como “channel=1-1” (si, es una porquería, pero es la forma que funciona actualmente, se aceptan parches).

Una vez que el archivo r2test.conf ha sido creado puedes ejecutar r2test de la siguiente manera:

```
# r2test -c r2test.conf
```

El comportamiento por default después de completar una llamada es poner el canal de audio en una aplicación tipo echo donde todo lo que se diga es repetido inmediatamente. Sin embargo, puedes especificar las siguientes opciones:

```
playaudio=yes  
audiofile=doc/intro-openr2-es.alaw
```

Asumiendo que estas ejecutando r2test en la misma carpeta donde tienes los fuentes, OpenR2 provee 3 audios de ejemplo en formato ALAW, intro-openr2-xx.alaw en es (español), br (portugués) y en (inglés). El formato ALAW es el único que entiende OpenR2.

Sólo hay un par de opciones más que quiero discutir, las opciones de debug. Son muy útiles cuando estas tratando de obtener ayuda de una lista de correo o de IRC. La salida de debug puede no significar nada para tí, pero ayuda mucho a cualquiera que entienda el protocolo de señalización R2.

La opción “loglevel” es poderosa para filtrar mensajes. Los siguientes valores están disponibles:

Los niveles 'error', 'warning', 'notice' y 'debug' se describen solos, Atenderá cualquier mensaje crítico, mensajes de peligro, noticias y mensajes generales de debug.

El nivel 'cas' hace referencia a los mensajes de rx (recepción) y tx (transmisión) de CAS.

El nivel 'mf' es para los mensajes de tonos multifrecuencia rx y tx.

El nivel 'all' es un valor especial que habilita todos los mensajes de debug.

El nivel 'nothing' es otro valor especial para no guardar ningún tipo de mensaje de debug.

Cuando solicites soporte, asegúrate que tu nivel de debug se encuentre en 'all'.

Puedes mezclar valores como “loglevel=warning,error,notice”.

Existe también otra opción interesante para debug que es independiente de la opción loglevel, la cual es “callfiles”. Esta opción acepta un “yes” o “no”. Si seleccionas “yes”, un archivo especial tipo “.call” se generará después de cada llamada en el directorio donde estés ejecutando r2test. Estos archivos “.call” contienen información muy valiosa para debug.

Las opciones descritas previamente son las más importantes, pero, no las únicas. Puedes encontrar documentación más extensa (y probablemente más completa) acerca de todas las

opciones en el archivo de prueba r2test.conf provisto en el directorio doc/ de la instalación OpenR2.

Hay una última opción para r2test, “-v”, que sirve para desplegar la versión instalada de OpenR2.

```
# r2test -v
```

Para más información acerca de r2test también puedes usar “man r2test” para ver la ayuda.

5. OpenR2 en Asterisk.

Asterisk no entiende señalización R2 por sí mismo, requiere de la ayuda de la librería OpenR2 para hacerlo. Pero también necesitas que Asterisk sepa de la presencia de la librería OpenR2, por lo que el código dentro de Asterisk pueda ser compilado para hacer uso de OpenR2. Actualmente hay 2 maneras de tener una versión de Asterisk con OpenR2 habilitado. Puedes descargar una versión SVN de Asterisk que ya se encuentre parchado y listo para usar OpenR2, o puedes descargar una versión normal de Asterisk y después descargar el parche OpenR2-Asterisk, y aplicarlo tú mismo.

Ten en cuenta, sin embargo, que si tu descargar una versión normal de Asterisk (.tar.gz) y luego intentas aplicar el parche OpenR2-Asterisk, el parche puede fallar si el tar.gz de Asterisk que uses es anterior a la última versión. Yo trato de tener los parches para la mayoría de las versiones mayores de Asterisk, pero las versiones menores pueden cambiar a menudo y no puedo siempre tener los parches actualizados tan rápido como los lanzan.

Asterisk 1.6 es una excepción, no tiene una rama estable como 1.2 y 1.4. La rama 1.6 es una copia del trunk, por lo que es una rama de desarrollo. Dados los ciclos de desarrollo de Asterisk, Asterisk 1.2 y Asterisk 1.4 nunca incluirán soporte para MFC/R2 en los lanzamientos oficiales, ya que esas dos versiones se encuentran “congeladas” en características, lo que significa que Digium, la compañía que soporta el desarrollo de Asterisk, NO aceptará cambios importantes de funcionalidad como soporte MFC/R2. Sin embargo, Asterisk 1.6 tiene muy buenas oportunidades de ser la primer versión de Asterisk en incluir soporte nativo a MFC/R2, y lo hará usando la librería OpenR2. Más información acerca de esto puede encontrarse aquí:

<http://bugs.digium.com/view.php?id=12509>

Si no quieres instalar Asterisk desde una rama SVN y prefieres aplicar el parche a una versión en específico de Asterisk puedes saltarte ésta sección.

== Instalación de Asterisk con una rama SVN MFCR2 ==

Habiendo dicho eso, tengo 3 ramas de Asterisk para soportar R2 en Asterisk 1.2, 1.4 y 1.6. Las ramas para Asterisk 1.2 y 1.4 son una copia de la última versión estable de Asterisk, yo solo copié el código usando SVN y agregué el código necesario para lograr que Asterisk use OpenR2. La rama 1.6 es una copia de la rama de desarrollo “trunk”, por lo tanto es considerado inestable. Puedes ayudar a acelerar el proceso de desarrollo probando ésta rama de desarrollo.

Asterisk-OpenR2 1.2: <http://svn.digium.com/svn/asterisk/team/moy/mfcr2-1.2>

Asterisk-OpenR2 1.4: <http://svn.digium.com/svn/asterisk/team/moy/mfcr2-1.4>

Asterisk-OpenR2 1.6: <http://svn.digium.com/svn/asterisk/team/moy/mfcr2>

Los pasos para instalar son los mismos que se usan para instalar cualquier versión de Asterisk, la única diferencia es que ésta vez descargas Asterisk usando SVN. Aquí encontrarás algunas instrucciones básicas dependiendo de la versión que quieras usar:

== Asterisk 1.2 ==

```
# svn checkout http://svn.digium.com/svn/asterisk/team/moy/mfcr2-1.2
# cd mfcr2-1.2
# make
# make install
```

Cuando ejecutas “make”, el script de instalación detectará si tienes OpenR2 instalado, si es así, compilará Asterisk con chan_zap.so listo para ser usado con MFC/R2. Si no tienes OpenR2 instalado, chan_zap.so compilará (si Zaptel 1.2 esta presente) pero **sin** soporte MFC/R2, que va en contra del propósito de los paquetes.

== Asterisk 1.4 ==

```
# svn checkout http://svn.digium.com/svn/asterisk/team/moy/mfcr2-1.4
# cd mfcr2-1.4
# ./configure --prefix=/usr
# make
# make install
```

Empezando con Asterisk 1.4, es necesario ejecutar el script de configuración para instalar Asterisk. Éste script de configuración hará el trabajo de detectar las librerías que tienes instaladas en tu sistema, incluyendo OpenR2. Si OpenR2 es encontrado, Asterisk será instalado con chan_zap.so habilitado para la señalización MFC/R2. Si no tienes OpenR2 instalado, chan_zap.so compilará **sin** soporte MFC/R2, que va en contra del propósito de los paquetes.

== Asterisk 1.6 ==

```
# svn checkout http://svn.digium.com/svn/asterisk/team/moy/mfcr2
# cd mfcr2
# ./configure --prefix=/usr
# make menuselect
```

Empezando con Asterisk 1.4, es necesario ejecutar el script de configuración para instalar Asterisk. Éste script de configuración hará el trabajo de detectar las librerías que tienes instaladas en tu sistema, incluyendo OpenR2. Si OpenR2 es encontrado, Asterisk será instalado con

chan_zap.so habilitado para la señalización MFC/R2. Si no tienes OpenR2 instalado, chan_zap.so compilará **sin** soporte MFC/R2, que va en contra del propósito de los paquetes.

Ésta es una rama de desarrollo, lo que significa que las cosas cambian frecuentemente in diferentes partes de Asterisk, no sólo en el código R2. Si quieres ayudar al soporte oficial de R2 en Asterisk, por favor prueba rama y déjanos saber tus resultados en <http://bugs.digium.com/view.php?id=12509>, esto ayuda a tener soporte R2 en lanzamientos oficiales de Asterisk de manera más pronta. Para instalar ésta rama **necesitarás tener instalado DAHDI**. La rama **no funciona con ninguna versión de Zaptel**, sólo con DAHDI, esto es por que Asterisk 1.6 quitó todo soporte para Zaptel.

Para verificar si chan_zap.so o chan_dahdi.so estan compilados propiamente y enlazados a OpenR2, puedes ejecutar el siguiente comando:

== Asterisk 1.2 y 1.4 ==

```
# ldd channels/chan_zap.so | grep openr2
```

== Asterisk 1.6 ==

```
# ldd channels/chan_dahdi.so | grep openr2
```

El comando deberá de mostrar algo parecido a:

```
libopenr2.so.1 => /usr/lib/libopenr2.so.1 (0x000000000589000)
```

La dirección de carga (0x000000000589000) y la versión de la librería (.1) pueden ser diferentes. Sin embargo, si no ves ninguna salida entonces OpenR2 no está habilitado en Asterisk, por lo tanto es posible que no hayas instalado debidamente OpenR2, o que lo hayas instalado DESPUES de haber instalado Asterisk.

== **Asterisk con parches OpenR2** ==

Un set de parches para diferentes versiones de Asterisk se encuentran disponibles en:

<http://code.google.com/p/openr2/downloads/list>

Los parches incluyen en el nombre el número de la versión más baja que saben que soportan. Por ejemplo, un parche llamado openr2-asterisk-1.4.18 sabe que funciona con Asterisk 1.4.18, y puede o no funcionar con Asterisk 1.4.18.1 o versiones más nuevas. Si un parche se puede aplicar sin ser rechazado por el comando “patch”, tiene muy buenas oportunidades de funcionar correctamente sin errores de compilación. Los parches no incluyen documentación o archivos de configuración de prueba, por lo que necesitarás descargar los archivos chan_dahdi_or_zapata.conf.sample.

En éste caso tenemos en cuenta ciertas versiones de Asterisk, pero puedes cambiar la versión a la que quieras teniendo en mente lo que mencioné acerca de las versiones de los parches.

== Asterisk 1.2 ==

```
# wget http://downloads.digium.com/pub/asterisk/asterisk-1.2.31.tar.gz
# tar -xzf asterisk-1.2.31.tar.gz
# cd asterisk-1.2.31
# wget http://openr2.googlecode.com/files/openr2-asterisk-1.2.30.3.patch
# patch -p0 < openr2-asterisk-1.2.30.3.patch
# make
# make install
```

== Asterisk 1.4 ==

```
# wget http://downloads.digium.com/pub/asterisk/asterisk-1.4.22.1.tar.gz
# tar -xzf asterisk-1.4.22.1.tar.gz
# cd asterisk-1.4.22.1
# wget http://openr2.googlecode.com/files/openr2-asterisk-1.4.22.patch
# patch -p0 < openr2-asterisk-1.4.22.patch
# ./bootstrap.sh
# ./configure --prefix=/usr
# make
# make install
```

Toma nota que es muy importante ejecutar el script “./bootstrap.sh” de Asterisk. El script bootstrap genera un nuevo script de configuración que detectará correctamente la presencia de OpenR2 y creará un Makefile que compie con chan_dahdi o chan_zap con soporte OpenR2. El script bootstrap parece ser muy sensible a la versión de “autoconf”. Si no quieres meterte en problemas te recomiendo que uses exactamente la versión 2.60 para autoconf, no mas nueva, no mas vieja. Si usas una versión diferente de autoconf el script bootstrap puede o no funcionar. Para revisar la versión de autoconf que tienes puedes ejecutar:

```
# autoconf --version
```

== Asterisk 1.6 ==

```
# wget http://downloads.digium.com/pub/asterisk/asterisk-1.6.0.3.tar.gz
# tar -xzf asterisk-1.4.22.1.tar.gz
# cd asterisk-1.4.22.1
# wget http://openr2.googlecode.com/files/openr2-asterisk-1.6.0.patch
# patch -p0 < openr2-asterisk-1.6.0.patch
# ./bootstrap.sh
# ./configure --prefix=/usr
# make
```

```
# make install
```

Consulta las notas de Asterisk 1.4 para más información de los requerimientos del script `./bootstrap.sh`.

Hasta este punto deberías tener Asterisk listo para soportar MFC/R2 con OpenR2. Para poder verificar que `chan_zap.so` o `chan_dahdi.so` están compilados debidamente y enlazados a OpenR2, puedes ejecutar el siguiente comando:

```
== Asterisk 1.2 and 1.4 ==
```

```
# ldd channels/chan_zap.so | grep openr2
```

```
== Asterisk 1.6 ==
```

```
# ldd channels/chan_dahdi.so | grep openr2
```

El comando debería de regresar algo como esto:

```
libopenr2.so.1 => /usr/lib/libopenr2.so.1 (0x0000000000589000)
```

La dirección de carga (0x0000000000589000) y la versión de la librería (.1) pueden ser diferentes. Sin embargo, si no ves ninguna salida entonces OpenR2 no está habilitado en Asterisk, por lo tanto es posible que no hayas instalado debidamente OpenR2, o que lo hayas instalado DESPUES de haber instalado Asterisk.

5.1 Configuración de Asterisk MFC/R2.

Cualquier rama que hayas instalado, podrás encontrar un archivo de configuración de ejemplo en `configs/chan_zap.conf.sample` o `configs/chan_dahdi.conf.sample`, revisa los parámetros que empiezan con 'mfcR2' para ver la documentación de cada uno de ellos. Si instalaste Asterisk con los parches de OpenR2 la documentación no esta incluida, por lo que puedes descargar el archivo de ejemplo desde la siguiente liga:

http://openr2.googlecode.com/files/chan_dahdi_or_zapata.conf.sample

La configuración de un enlace R2 varia dependiendo el país de instalación o el modelo del PBX al que tratas de conectar. El paquete OpenR2 incluye un directorio `doc/` con un subdirectorio `asterisk/` con configuraciones de ejemplo para varios países. Revisa los archivos en dicha carpeta para ver si tu país tiene ya ejemplos de configuración.

6. OpenR2 en FreeSwitch.

Aún no soportado :-(... pero ya viene!

7. Construyendo paquetes.

7.1 Construyendo paquetes para Debian

Descarga openr2.tar.gz y descomprímelo. Después entra al directorio openr2 y escribe el siguiente comando:

```
# dpkg-buildpackage -uc -us
```

El paquete se generará en el directorio superior.

7.2 Construyendo paquetes RPM.

Download the openr2 .tar.gz and decompress it. Then change into the openr2 directory and type this command:

```
# rpmbuild -ta openr2-1.0.0.tar.gz
```

Los paquetes serán generados en el directorio rpmbuilds/RPMS/<arquitectura>