

O Guia da OpenR2 – Janeiro 2009 - Versão 0.2-BR

Moisés Silva <moy@sangoma.com>
Alexandre Alencar <alexandre.alencar@gmail.com>

<http://www.libopenr2.org/>
<http://code.google.com/p/openr2/>

NOTA DE TRADUÇÃO: Esta tradução não foi extensamente revisada e é provável que existam erros, caso encontre algum, envie-nos um e-mail sugerindo a correção.

1. Sobre este guia

A biblioteca OpenR2 implementa o protocolo de sinalização MFC/R2 sobre linhas E1 usando a *Zapata Telephony Interface*, *DAHDI* e em um futuro próximo a biblioteca de abstração *TDM OpenZAP*. Este documento NÃO descreve as interfaces de programação providas pela OpenR2, por favor, utilize o código como referência para isto :-), e não hesite em perguntar-me qualquer dúvida que você tenha.

Este documento descreve a instalação da biblioteca para ser utilizada pelo *Asterisk* e/ou *FreeSwitch*.

Para questões relacionadas à instalação ou problemas de desenvolvimento, **por favor, não me contate diretamente**, a menos que você deseje me pagar para dar-lhe suporte :-). Ao invés disto, use a lista de discussão Digium's asterisk-r2 em: <http://lists.digium.com/mailman/listinfo/asterisk-r2>, ou entre no IRC em irc.freenode.org no canal *#openr2*. Eu geralmente estou lá com o *nick 'moy'*.

Se há alguma variação MFC/R2 que não é suportada e você precisa (ou não está funcionando como esperado), envie-me um *e-mail*. Eu também apreciaria qualquer contribuição ou correção para este documento ou código da biblioteca.

2. O que é MFC/R2?

O MFC/R2 é uma sinalização telefônica largamente utilizada no México, Brasil e outros países na América Latina e Ásia. MFC/R2 significa *Multi Frequency Compelled R2*. Comparado a protocolos de sinalização mais recentes como ISDN PRI/BRI ou SS7, o R2 oferece um conjunto limitado de funcionalidades. A sinalização é utilizada apenas para estabelecer e terminar a chamada. Algumas das variações MFC/R2 podem enviar pulsos de tarifação durante a chamada, mas estes são raramente utilizados.

Há versões analógicas e digitais do MFC/R2, qualquer referência a MFC/R2 ou R2 neste documento refere-se à versão digital que usa facilidades E1. A maioria dos leitores não estarão interessados na versão analógica porque não é mais muito utilizada. O MFC/R2 é definido pelo ITU, muito embora a maioria dos países utilizando o MFC/R2 não sigam a especificação do ITU e implementem sua própria variação nacional do padrão.

3. Como o MFC/R2 funciona?

Seja você um programador que irá utilizar a biblioteca ou um administrador de sistemas que vai instalá-la junto com um *software* de telefonia (ou seja, *Asterisk* ou *FreeSwitch*), é uma boa idéia de ter um conhecimento básico de como funciona a sinalização, o que permitirá que você solucione problemas mais facilmente. Se você não estiver interessado

nisto e só quer um guia rápido para instalar a OpenR2 e tê-la funcionando com seu *software* de telefonia, você pode pular esta seção.

O MFC/R2 é um protocolo de sinalização *peer-to-peer*, o que significa que existem apenas duas partes envolvidas em um *link* R2 E1 e ambas as partes se comportam da mesma maneira, ao contrário do PRI quando você tem as faces "NET" e "CPE" do link.

Como mencionado anteriormente, o MFC/R2 significa *Multi Frequency Compelled R2*. O nome descreve a natureza desta sinalização, onde você tem 2 tipos de sinais:

Sinais de linha são usados para monitorar o estado da chamada e sinais MF são usados para transmitir informações da chamada durante a configuração da chamada (DNIS, ANI, *Calling Party Category*). Sinais de linha são transmitidos através de sinais CAS que viajam através do canal 16 do *link* E1. Toda a sinalização CAS para cada canal do E1 é multiplexada através deste canal. A cada 2ms cada lado da ligação atualiza seus 4 *bits* de sinais CAS conhecidos como *bits* ABCD.

O MFC/R2 usa 2 desses 4 bits para enviar os seguintes sinais: *Idle, Block, Seize, Seize Ack, Clear Back, Forced Release, Clear Forward, Answer*. Apenas 2 *bits* para 7 possíveis sinais não é possível, sem repetir um padrão de *bit*, pois alguns dos sinais destes 7 sinais R2 têm o mesmo padrão de *bit*, mas esse não é um problema, considerando que você não pode ir de *Idle* a *Forced Release* por exemplo, por isso, mesmo quando o padrão de *bit* para *Forced Release* e *Seize* são os mesmos, a pilha de protocolo sabe o que o outro extremo da ligação quer. A razão para usar apenas 2 *bits* com 4 disponíveis é histórica e vem desde os tempos quando a versão analógica do MFC/R2 foi portada para trabalhar em um mundo digital.

A tabela a seguir descreve os padrões de *bits* utilizados na sinalização R2 através dos *bits* CAS ABCD.

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

Por outro lado, para **sinais de endereços**, são 15 diferentes sinais MF, que são tons sonoros composto de 2 frequências que viajam utilizando o canal de áudio em si, é por isso que um detector de áudio é um componente importante de uma pilha R2, a OpenR2 por padrão utiliza seu próprio detector R2 MF, emprestado da biblioteca SpanDSP de Steve Underwood. Você não precisa instalar a SpanDSP para utilizar a OpenR2, uma vez que o detector é "*built-in*".

O ITU define quais frequências podem ser mixadas para compor os tons MF e atribui significados a estes tons. No entanto, alguns países atribuem diferentes significados a estes tons MF. Os tons MF são identificados na biblioteca OpenR2 utilizando os números de 1 a 0 (0 a 10) e letras B a F (A não é utilizado, 0 é usado em seu lugar). Se ativar o registo detalhado para a pilha OpenR2 você terá os detalhes de tons, que são enviados e recebidos durante cada configuração de chamada. Mais informações sobre este assunto na seção "Solução de problemas" ou na seção instalação de sua plataforma de telefonia favorita.

Os tons MF são utilizados para transmitir ANI (*Automatic Identification Number*, vulgarmente conhecido por alguns usuários como *Caller ID*), DNIS (*Dialed Number Identification Service*, ou seja, o número discado ou o número de destino) e *Calling Party Category*. Provavelmente outras informações são trocadas, bem como nas ligações internacionais R2 (mas não tenho experiência com estes, e não são provavelmente mais utilizados). Tão logo a chamada seja aceita ou rejeitada, o detector MF é desligado e outros sinais MF não serão trocados, o canal de áudio, então pode ser usado para voz ou mídia de início.

4. Instalação Zaptel/DAHDI.

A instalação da OpenR2 é bastante simples, porém, existem alguns pré-requisitos que devem ser atendidos. A partir de agora, a biblioteca exige o Zaptel 1.2/1.4 ou DAHDI para compilar. O *script* configure vai detectar se você tem o Zaptel 1.2, Zaptel 1.4 ou DAHDI instalado. Você pode enfrentar problemas se tiver mais de um desses pacotes instalados porque a OpenR2 irá compilar usando o primeiro que encontrar.

Zaptel 1.2 instala o *header* zaptel.h em /usr/include/linux/
Zaptel 1.4 instala o *header* zaptel.h em /usr/include/zaptel/
DAHDI instala o *header* user.h em /usr/include/dahdi/

Os caminhos de instalação podem variar, mas é uma boa idéia ter a certeza que só tem um desses. Se você não tem Zaptel ou DAHDI instalado, vá para <http://downloads.digium.com/> para obtê-lo.

Se você planeja instalar o Asterisk 1.2, você irá precisar do Zaptel 1.2
Se você planeja instalar o Asterisk 1.4, você irá precisar do Zaptel 1.4 ou DAHDI.
Se você planeja instalar o Asterisk 1.6, você irá precisar do DAHDI.

== **Zaptel 1.2** ==

```
# wget http://downloads.digium.com/<zaptel-1.2-release-path-to-file>.tar.gz  
# tar -xvpzf zaptel-<release>tar.gz  
# cd zaptel-<release>  
# make  
# make install
```

== **Zaptel 1.4** ==

```
# wget http://downloads.digium.com/<zaptel-1.4-release-path-to-file>.tar.gz  
# tar -xvpzf zaptel-<release>tar.gz  
# cd zaptel-<release>  
# make  
# make install
```

== DAHDI ==

Com o lançamento do DAHDI, a Digium dividiu as ferramentas de usuário e *drivers* de *kernel* em 2 pacotes: dahdi-tools e dahdi-linux.

Você deve instalar primeiro dahdi-linux, que contém os próprios drivers, antes do dahdi-tools.

```
# wget http://downloads.digium.com/<dahdi-linux-path-to-file>.tar.gz
# tar -xvpzf dahdi-linux-<version>.tar.gz
# cd dahdi-linux-<version>
# make
# make install
```

Agora você deve baixar e instalar o dahdi-tools. A OpenR2 não requer necessariamente o dahdi-tools, no entanto o dahdi-tools fornece vários binários que são úteis para a resolução de problemas (como o substituto do zttool chamado dahdi_tool).

```
# wget http://downloads.digium.com/<dahdi-tools-path-to-file>.tar.gz
# tar -xvpzf dahdi-tools-<version>.tar.gz
# cd dahdi-tools-<version>
# ./configure --prefix=/usr
# make
# make install
```

Agora, se você tiver uma placa Sangoma, você precisará instalar o wanpipe. Os drivers Sangoma complementam o Zaptel / DAHDI, mas precisamos instalá-los primeiro. Os drivers Sangoma estão incluídos em um pacote conhecido como wanpipe. Consulte a wiki da Sangoma (<http://wiki.sangoma.com/>) para obter informações sobre instalação.

Agora que você tem o Zaptel ou DAHDI instalado, precisará configurar o hardware. O DAHDI é configurado no arquivo `/etc/dahdi/system.conf` e o Zaptel é configurado em `/etc/zaptel.conf`. Felizmente para nós, `/etc/dahdi/system.conf` tem compatibilidade com a sintaxe antiga (mas suporta novas opções). Lembre-se que no mundo R2, a ligação física é E1 não T1 (pelo menos eu nunca vi um R2 instalação com alguma outra coisa do que um E1), se você recebeu suas placas dos EUA ou Canadá provavelmente eles enviaram configuradas para T1. As placas Sangoma podem ser alteradas via software, porém a configuração de placas Digium T1/E1 precisam ser mudadas usando jumpers na placa. A melhor maneira de determinar se o span está em modalidade E1 ou T1 é lendo o conteúdo do arquivo `/proc/zaptel/<spanno>` ou `/proc/dahdi/<spanno>` com o comando 'cat', tal como segue:

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

ou se você tem o DAHDI.

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

Isto irá mostrar a configuração para o span 1, para um E1 isto irá mostrar 31 chanais.

Um span E1 DAHDI ou Zaptel é configurado como segue:

```
span=1,1,0,cas,hdb3
cas=1-15:1101
dchan=16
```

cas=17-31:1101

É importante para você saber o significado de cada parâmetro. Descrevo brevemente cada um, mas o encorajo a ler o `zaptel.conf.sample` ou `system.conf.sample` na sua distribuição Zaptel / DAHDI.

As linhas "span" determinam, naturalmente, a configuração do span. Você pode ver um span como um grupo de canais. Você pode ver quantos spans são registrados pelo conteúdo do diretório `/proc/zaptel` ou `/proc/dahdi`. Cada arquivo listado no diretório representa um span, quando o span representa um grupo de canais de hardware ou uma canal emulado via software depende do tipo de span. Os módulos de Kernel como o `ztdummy` registram um span "dummy" apenas para fornecer um relógio (utilizando as facilidades de timer do kernel). Ok, é o suficiente sobre spans, basta lembrar que se você quiser configurar um dos spans listados em `/proc/zaptel` ou `/proc/dahdi` você terá que fazê-lo em `/etc/zaptel.conf` ou `/etc/dahdi/system.conf` utilizando o parâmetro `span`. O parâmetro `span` é composto pelos seguintes elementos:

```
span=<span      num>,<timing      source>,<line      build      out
(LBO)>,<framing>,<coding>[,crc4|yellow[, yellow]]
```

Para efeito de linhas R2, sinta-se livre para ignorar qualquer coisa além do valor `<coding>`. Vamos descrever os outros.

`` refere-se a um número inteiro que representa o número do span. No nosso exemplo estamos configurando o span 1.

`<timing source>` refere-se a um inteiro que determina se a outra ponta (qualquer que esteja conectada do outro lado) vai fornecer o sinal do relógio e será utilizado como fonte principal de relógio. Se você estiver conectado à PSTN (sua operadora de telefonia) é mais provável que você precise definir isso para um número maior que 0, dependendo de quantos spans você tenha. Por exemplo, se você tem 2 spans ligados à PSTN um desses spans devem ser fixados com `<timing source>` igual a 1 e os outros igual a 2, marcando-as como a primeira fonte de relógio mestre e segunda fonte de relógio relógio. Se você tem o span ligado a algum PBX legado, provavelmente, o PBX legado espera receber clocking da PSTN, mas, uma vez que não está conectado à PSTN, mas ao seu servidor, você deve fornecer o relógio ao PBX, neste caso você definir este valor para 0.

span=1,0,0,cas,hdb3

Isso significa que o span 1 irá fornecer o relógio para o que estiver ligado ao outro extremo. Esteja ciente de que o arquivo `zaptel.conf.sample` indica claramente que sincronização incorreta de temporização pode causar cliques / ruído no áudio, a má qualidade ou faxes que falham e mau funcionamento de modem entre outros problemas.

`<line build out (LBO)>` é um inteiro entre 0 e 7, que determina o nível de transmissão do span, você pode escolher esse valor, de acordo com a tabela apresentada no arquivo de exemplo de configuração do 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> e <coding> determinam a sinalização e formatação de baixo nível de quadros E1, para a sinalização R2 você irá utilizar CAS e hdb3 respectivamente.

Era só o que faltava para o "span". Agora, vamos ver o que as linhas "CAS" significam. Após a configuração do span, é preciso configurar cada canal do span com a devida sinalização. Neste caso, o span CAS terá todos os seus canais com sinalização CAS (também conhecido como sinalização de usuário).

```
cas=x-y:1101
```

Isso significa que a faixa de canais de x a y irá reportar bits CAS. O :1101 é a posição inicial dos bits CAS (também conhecidos como bits ABCD), que são utilizados pelo R2 para sinalizar o estado da linha (answer, hangup etc.) O sinal 1101 significa "Blocked" na R2, o que significa que quando o hardware acorda pela primeira vez não será capaz de receber chamadas (o Asterisk ou qualquer outro software usando R2 irá alterar estes bits para 1001, o que significa IDLE, ou pronto para fazer chamadas).

O parâmetro dchan = 16 tem sido um pouco controverso, dado que algumas pessoas não o definem, enquanto outros o fazem. O canal 16 (provavelmente melhor dito, timeslot 16) é usado para transportar a multiplexagem de bits CAS para os outros canais E1. Na minha experiência os bits são recebidos, queira você defina ou não o parâmetro dchan = 16, mas configurá-lo contribui para a definição clara de alguma sinalização anterior de canal pode ter sido feita. Recomendo a definição, a menos que você tenha problemas na execução do "ztcfg" ou "dahdi_cfg". Algumas versões do ztcfg não vão aceitar um span CAS com dchan = 16 e se recusará a configurar os canais, basta remover o parâmetro para solucioná-lo.

4.1 Instalação do Sangoma Wanpipe.

Essa etapa é necessária somente se você possuir hardware Sangoma. Uma vez que o hardware Sangoma funciona em diferentes cenários, é necessário alguns passos extras de instalação para integrar plenamente as funcionalidade de placas Sangoma com a infraestrutura de kernel do Zaptel / DAHDI .

Informações sobre a instalação da Sangoma Wanpipe podem ser encontradas aqui:

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

5. OpenR2

Agora que temos a configuração do hardware no lugar, podemos proceder para o download e instalação da OpenR2. O site oficial para a OpenR2 é <http://www.libopenr2.org/>, no entanto, o código fonte está hospedado no site Google Code <http://code.google.com/p/openr2/> e a seção downloads está em <http://code.google.com/p/openr2/downloads/list>

1. Faça Download da OpenR2 da página do 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. Execute o script 'configure'.

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

A opção `--prefix=/usr` irá instalar a biblioteca no diretório `/usr/lib/`, mas se você não especificar a opção `--prefix` será instalado no diretório `/usr/local/`, eu recomendo instalar com `--prefix=/usr`, acredite em mim, vai livrá-lo de problemas (na maioria das vezes). Se o script `configure` não encontrar os cabeçalhos Zaptel ou DAHDI então ele irá falhar com um erro. Certifique-se de pesquisar por erros no `configure` e instalar qualquer dependência que possa estar faltando. Em geral, apenas o Zaptel ou DAHDI é necessário.

3. Compile.

```
# make
```

Se não houver erros neste passo, continue. Um erro nesta fase muito provavelmente significa um bug, ou que você está compilando em uma plataforma não suportada. Você pode buscar por ajuda nas listas ou IRC (v. 1. Sobre este guia). Mas não se esqueça de procurar por uma descrição do erro na saída e fornecê-la.

4. Instale

```
# make install
```

É isso, você deve estar pronto para seguir. Lembre-se que, geralmente, você precisa ser `root` para executar esta última etapa, a menos que você tenha especificado a opção `--prefix` para o script apontando para um diretório que o usuário tenha direito de escrita.

Agora que a biblioteca está instalada, você pode proceder para instalar o Asterisk. No entanto, você também pode testar suas linhas R2 sem o Asterisk estar envolvidos. Leia a seção 6 "Teste a OpenR2" para saber como fazer o teste sem Asterisk. Se você não está interessado em testar sem o Asterisk, pode avançar para o item 6 "OpenR2 em Asterisk".

5.1. Teste a OpenR2

A biblioteca OpenR2 provê um pequeno programa de teste chamado `r2test` (a menos que você especifique a opção `--without-r2test`). É uma pequena aplicação que utiliza um arquivo de configuração para "escutar" por chamadas nos dispositivos Zaptel/DAHDI que você especificar e atender às chamadas, pondo-as em uma aplicação de eco ou reproduzir um arquivo (em formato `alaw`). O programa `r2test` não apenas espera por chamadas, mas ele também pode ser configurado para fazer chamadas.

Para usar o `r2test` você precisa primeiro criar um arquivo de configuração. Um exemplo está incluído no arquivo de configuração da OpenR2 no diretório `doc/r2test.conf`, o formato do arquivo é:

```
# isto é um comentário  
parâmetro1=valor  
parâmetro2=valor  
channel=início-fim
```

Quando o `r2test` encontra um parâmetro `channel`, ele irá criar um novo bloco de canais DAHDI/Zaptel começando com "início" e terminando com "fim" e configurá-los com

quaisquer parâmetros precedentes. Você deve certificar-se que "inicio" e "fim" são dispositivos DAHDI / Zaptel válidos. Por exemplo, esta é uma configuração mínima para o arquivo r2test.conf:

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

Um arquivo r2test.conf com tal conteúdo teria aberto os canais 1-15 do DAHDI/Zaptel, e configurado-os com caller=no, maxani=10 e maxdnis=4. Se você receber um erro "*No such file or directory*", pode ser que você não tenha carregado os *drivers* DAHDI/Zaptel. Quando os *drivers* DAHDI/Zaptel são carregados, eles criam dispositivos em /dev/dahdi ou /dev/zap, no exemplo de configuração anterior, os dispositivos /dev/zap/1 a /dev/zap/15 ou /dev/dahdi/1 a /dev/dahdi/15 são necessários.

O parâmetro "*variant*" determina qual variante R2 será usada. Você pode listar as variantes suportadas com o comando:

```
# r2test -l
```

Para a OpenR2 1.0.0 a lista é:

```
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
```

O parâmetro "*caller*" determina se o grupo de canais irá fazer chamadas ou esperar por chamadas, "*caller=yes*", significa que os canais irão fazer chamadas, neste caso o parâmetro "*dnid*" deve ser especificado.

Os parâmetros "*maxani*" e "*maxdnis*" simplesmente determinam o qual é o número máximo de dígitos que serão recebidos para ANI e DNIS.

O parâmetro "*channel*" *deve* estar no formato "x-y" para especificar a intervalo de canais para os quais os parâmetros de configuração anteriores serão aplicados (como no zapata.conf ou chan_dahdi.conf). Se você só precisa de 1 canal você deve especificá-lo como "channel=1-1" (sim, é chato, mas é a forma como atualmente funciona, patches são aceitos).

Depois que o arquivo r2test.conf for criado, pode executar o r2test como no exemplo:

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

O comportamento padrão após completar uma chamada é por o áudio em uma aplicação de eco onde tudo é dito é repetido imediatamente. No entanto, você também pode especificar as opções:

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

Supondo que você está executando o r2test a partir da mesma pasta onde você tem os fontes. A OpenR2 fornece 3 exemplos de arquivos de áudio no formato ALAW, intro-openr2-xx.alaw em es (espanhol), br (Português) e en (Inglês). O formato ALAW é o único que a OpenR2 entende.

Existe apenas um par de opções mais que eu quero discutir. Opções de debugging. Elas são úteis quando você está tentando buscar ajuda em listas ou IRC. Saídas de depuração pode não significar nada para você, mas vai ajudar muito a quem quer que compreenda o protocolo de sinalização R2.

A opção "LOGLEVEL" é bastante poderosa para filtrar mensagens. Os seguintes valores de *logging* estão disponíveis:

Níveis 'error', 'warning', 'notice' e 'debug' são auto-descritivos, irão registrar qualquer mensagem crítica, de alerta, aviso e depuração geral.

Nível 'cas' é para registrar sinais CAS tx e rx.

Nível 'mf' é para registrar os tons multifrequênciais tx e rx.

Nível 'all' é um valor especial que ativa todas as mensagens de depuração.

Nível 'nothing' é outro valor especial que desativa todas as mensagens.

Quando for solicitar suporte certifique-se que o seu nível de depuração está definido como 'all'.

Você pode combinar valores como: "loglevel=warning,error,notice".

Há outra opção interessante para a depuração que é independente da opção loglevel, esta é a "callfiles". Esta opção aceita "yes" ou "no". Se você escolher "yes" um arquivo ".call" especial será criado após cada chamada, no diretório onde você executar o r2test. Estes arquivos ".call" contêm valiosas informações de depuração.

As opções descritas anteriormente, são as mais importantes, mas não são as únicas. Você pode encontrar documentação mais extensa (e provavelmente mais exata) sobre todas as opções no arquivo de exemplo r2test.conf fornecido com a OpenR2 na pasta doc/.

Há apenas uma opção final para o r2test, que é "-v" para mostrar a versão instalada OpenR2.

```
# r2test -v
```

Para obter mais informações sobre o r2test você pode tentar também "man r2test" para ver a ajuda.

6. OpenR2 no Asterisk.

O Asterisk por si só não entende a sinalização R2, requer a ajuda da biblioteca OpenR2 para fazê-lo. Mas você também precisa dar ciência ao Asterisk da presença da biblioteca OpenR2 de forma que o código pode ser compilado de forma adequada dentro do Asterisk para fazer uso da OpenR2. Neste momento, existem 2 formas de obter uma versão do Asterisk com a R2 habilitada. Você pode baixar uma versão de um *branch* SVN do Asterisk que já está pronta para funcionar com a OpenR2, ou você pode baixar uma versão normal do asterisk.tar.gz e depois fazer o download do patch OpenR2 para Asterisk e aplicá-los.

Esteja ciente, no entanto, que, se você baixar uma versão normal do asterisk.tar.gz e tentar aplicar o patch OpenR2 para Asterisk, o patch pode falhar se aplicar a um asterisk.tar.gz muito antigo ou muito recente. Eu tentarei disponibilizar patches para as principais versões do Asterisk, mas as versões mudam com muita frequência e eu nem sempre poderei acompanhar essas mudanças rapidamente.

O Asterisk 1.6 é uma exceção, por não ter um *branch* estável como no 1.2 e 1.4. O *branch* 1.6 é uma cópia do *trunk* e, portanto, é um *branch* de desenvolvimento.

Dado os ciclos de desenvolvimento do Asterisk, o Asterisk 1.2 e Asterisk 1.4 nunca incluirão suporte a MFC/R2 nos lançamentos oficiais porque essas 2 versões já estão congeladas para novas funções, o que significa que a Digium, empresa líder do desenvolvimento do Asterisk, não irá aceitar grandes mudanças de funcionalidades como suporte a MFC/R2. No entanto, o Asterisk 1.6 tem boas chances de ser a primeira versão do Asterisk a incluir suporte nativo a MFC/R2, e vai fazê-lo utilizando a biblioteca OpenR2. Mais informações sobre o andamento dessa inclusão podem ser encontradas aqui:

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

Se você não quiser instalar a partir de um *branch* SVN do Asterisk e você preferir aplicar um patch para uma versão específica do Asterisk você pode pular esta seção.

== Instalação do Asterisk com MFCR2 de um *branch* SVN ==

Dito isto, tenho 3 *branches* do Asterisk, a fim de suportar a R2 no Asterisk 1.2, 1.4 e 1.6. Os *branches* para Asterisk 1.2 e 1.4 são uma cópia da última versão estável do Asterisk, eu apenas faço uma cópia do código usando o SVN e acrescentou o código necessário para fazer uso da OpenR2 no Asterisk. O *branch* 1.6 é uma cópia do "*trunk*" *branch* de desenvolvimento e, portanto, é considerado instável, você pode ajudar a acelerar o processo de desenvolvimento, testando a evolução deste *branch*.

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

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

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

Os passos para instalação são os mesmos que geralmente segue para instalar qualquer versão do Asterisk, a única diferença é que desta vez você baixar o Asterisk usando o SVN. Aqui estão algumas breves instruções, dependendo versão do Asterisk que você deseja.

== Asterisk 1.2 ==

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

Quando você digitar "make", o script *install* irá detectar se você já tem a OpenR2 instalada, em caso afirmativo, ele irá compilar o Asterisk com o *chan_zap.so* pronto para ser usado com MFC/R2. Se você não tiver a OpenR2 instalada, o *chan_zap.so* ainda pode ser compilado (se o Zaptel 1.2 estiver presente), mas **sem** suporte a MFC/R2, o que frustra a finalidade de todo o *branch*, de qualquer forma.

== 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
```

Começando com o Asterisk 1.4, é necessário executar o script *configure* para instalar o Asterisk. O script *configure* irá cuidar de detectar as bibliotecas que você tem instaladas no seu sistema, incluindo a OpenR2. Se a OpenR2 for encontrada, o Asterisk será instalado com o *chan_zap.so* ativado para a sinalização MFC/R2. Se você não possuir a OpenR2 instalada, o *chan_zap.so* ainda pode ser compilado (se o Zaptel 1.4 estiver presente), mas **sem** suporte a MFC/R2, o que frustra a finalidade de todo o *branch*, de qualquer forma.

== Asterisk 1.6 ==

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

Começando com o Asterisk 1.6, é necessário executar o script *configure* para instalar o Asterisk. O script *configure* irá cuidar de detectar as bibliotecas que você tem instaladas no seu sistema, incluindo a OpenR2. Se a OpenR2 for encontrada, o Asterisk será instalado com o *chan_dahdi.so* ativado para a sinalização MFC/R2. Se você não possuir a OpenR2 instalada, o *chan_dahdi.so* ainda pode ser compilado (se o DAHDI estiver presente), mas **sem** suporte a MFC/R2, o que frustra a finalidade de todo o *branch*, de qualquer forma.

Este é um *branch* de desenvolvimento, o que significa que as coisas podem mudar frequentemente em diferentes partes do Asterisk, e não apenas o código R2. Se você quiser contribuir para apoiar a R2 oficialmente no Asterisk, por favor teste este *branch* e deixe-nos saber seus resultados em <http://bugs.digium.com/view.php?id=12509>, isso irá ajudar a ter o suporte a R2 nas versões oficiais do Asterisk mais cedo. Para instalar este *branch* você irá precisar do DAHDI instalado. Este *branch* não funciona com nenhuma versão do Zaptel, apenas com DAHDI, isto porque o suporte ao Zaptel foi removido completamente do Asterisk 1.6.

A fim de verificar se *chan_zap.so* ou *chan_dahdi.so* estão devidamente compilados e relacionados com a OpenR2, você pode executar este comando:

== Asterisk 1.2 e 1.4 com Zaptel ==

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

== Asterisk 1.4 e 1.6 com DAHDI ==

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

Este comando deverá apresentar algo como:

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

O endereço de carga (0x000000000589000) e a versão da biblioteca (.1) podem ser diferentes. No entanto, se você não vê nenhuma saída, então a OpenR2 não está habilitada no Asterisk, portanto é muito provável que você não instalou corretamente a OpenR2 ou a instalou depois de ter instalado o Asterisk.

== Asterisk com patches da OpenR2 ==

Um conjunto de *patches* para diferentes versões Asterisk estão disponíveis em:

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

Os *patches* incluem em seus nomes a menor versão do Asterisk para as quais seu funcionamento é conhecido. Por exemplo, um *patch* nomeado como openr2-asterisk-1.4.18 tem funcionamento conhecida para no mínimo, com o Asterisk 1.4.18, pode ou não funcionar com a Asterisk 1.4.18.1 ou versões superiores. Se um *patch* for aplicado sem haver rejeições pelo utilitário "patch", há boas chances de funcionar bem, sem ocorrerem erros. Os *patches* não incluem documentação com exemplos de configuração, de forma que você precisará fazer o download do arquivo chan_dahdi_or_zapata.conf.sample, a fim de ler a documentação de exemplo.

Neste caso, nós não assumimos determinadas versões do Asterisk, mas você pode mudar a versão para a qual desejar desde que você tenha em mente o que eu já referi em relação a versões dos *patches*.

== 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
```

Por favor, note que executar o script "./bootstrap.sh" do Asterisk é muito importante. O script *bootstrap* gera um novo script que irá configurar corretamente e detectar a presença da OpenR2, e criar um *makefile* que compilará o chan_dahdi ou chan_zap com o suporte a

OpenR2. O script *bootstrap* parece ser muito sensível à versão do "autoconf". Se você não quiser ocorrer em problemas é recomendado que você use exatamente a versão 2.60 do autoconf, e não as mais novas, e nem mais velhas. Se você usar uma versão diferente do autoconf então o script *bootstrap* pode ou não funcionar. Para verificar a versão que você possui, pode executar o autoconf:

```
# autoconf --version
```

== Asterisk 1.6 ==

```
# wget http://downloads.digium.com/pub/asterisk/asterisk-1.6.0.3.tar.gz
# tar -xzf asterisk-1.6.0.3.tar.gz
# cd asterisk-1.6.0.3
# 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
```

Veja as notas sobre o Asterisk 1.4 para obter mais informações sobre os requisitos do script *./bootstrap.sh*.

Neste ponto você deve ter o Asterisk pronto com suporte a MFC/R2 com a OpenR2. A fim de verificar se o *chan_zap.so* ou *chan_dahdi.so* estão devidamente compilados e relacionados com a OpenR2, você pode executar este comando:

== Asterisk 1.2 and 1.4 ==

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

== Asterisk 1.6 ==

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

Este comando deverá apresentar algo como:

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

O endereço de carga (0x000000000589000) e a versão da biblioteca (.1) podem ser diferentes. No entanto, se você não vê nenhuma saída, então a OpenR2 não está habilitada no Asterisk, portanto é muito provável que você não instalou corretamente a OpenR2 ou a instalou depois de ter instalado o Asterisk.

6.1. Asterisk MFC/R2 configuration.

Qualquer que seja o *branch* que você instalou, um arquivo de configuração de exemplo é fornecido, em *configs/chan_zap.conf.sample* ou *configs/chan_dahdi.conf.sample*, pesquise pelos parâmetros começando com 'mfc2' para ver a documentação de cada um desses. Se o Asterisk com a OpenR2 foi instalado usando o patches, então a documentação não está incluída e você pode baixar aqui um exemplo de configuração:

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

A configuração de um link R2 irá variar dependendo do seu país ou do modelo de PABX ao qual você está tentando conectar-se. O pacote da OpenR2 inclui um diretório doc/ com um subdirectório asterisk/ contendo exemplos de configurações para vários países. Dê uma olhada lá para ver se já existem alguns exemplos para o seu país.

7. OpenR2 in FreeSwitch.

Não há suporte ainda :(... mas está chegando!

8. Construindo pacotes

8.1 Construindo pacotes Debian

Faça o Download do arquivo openr2.tar.gz e descompacte-o. Entre no diretório openr2 e digite o seguinte comando:

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

Os pacotes serão gerados no diretório anterior.

8.2 Construindo pacotes RPM

Faça o Download do arquivo openr2.tar.gz e digite o seguinte comando:

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

O pacote será gerado em rpmbuilds/RPMS/<arch>