

Mi vida con djbdns

Henning Brauer <lists-lwd@bsws.de>, Trad. Iván Juanes Prieto <kerberos@inslfug.org> 8 January 2002

Contents

1	Introducción	3
2	Otros recursos	3
3	Imagen de conjunto	3
3.1	Resolutores y servidores de nombres: definiciones	4
3.2	Tipos de consultas DNS	4
3.3	Estructura de los nombres de dominio	4
3.3.1	Tipos comunes de registros de recursos (RRs)	5
3.3.2	Ejemplo: encontrar la dirección IP de www.example.com	5
4	Los componentes de djbdns: cómo interactúan	6
4.1	dnscache	6
4.2	tinydns y tinydns-data	7
4.3	axfrdns	7
4.4	axfr-get	8
4.5	El programa cliente de DNS	8
4.6	dnsfilter	9
4.7	Servidores especializados	10
4.8	daemontools	10
4.9	ucspi-tcp	11
5	Configuraciones de ejemplo	11
5.1	Configuración sencilla	11
5.1.1	Configuración de dnscache	12
5.1.2	Configuración de tinydns	12
5.1.3	Configuración de axfrdns	13
5.2	split horizon	13
5.3	DNS secundarios	13
5.3.1	tinydns a tinydns	13
5.3.2	Transferencia de zonas de BIND a tinydns	14

5.3.3	Transferencia de zonas de tinydns a BIND	14
5.4	Sitios grandes con necesidad de ficheros separados para cada zona	16
5.5	Uso de djbdns en entornos de ISP	16
5.5.1	Exportar tinydns-data desde una base de datos relacional (RDBMS)	16
5.5.2	Ejecución de servidores tinydns adicionales	19
5.5.3	Conclusión	19
5.6	Servidor raíz de nombres privado	19
5.7	Servidores raíz alternativos	20
5.8	Servir los mismos datos a múltiples interfaces	20
6	Mantenimiento	21
6.1	tinydns	21
6.2	dnscache	22
7	Comparación con BIND; cuestiones relacionadas con la migración	22
7.1	Migración	22
7.1.1	Conversión de sus ficheros de zona	22
7.1.2	Si es Usted servidor primario de un servidor BIND	22
8	Comprobación de configuraciones DNS y diagnóstico de problemas	23
9	Rendimiento	24
9.1	uso de memoria de dnscache	24
10	Instalación de paquetes	24
10.1	Instalación manual de fuentes	24
10.1.1	daemontools	24
10.1.2	ucspi-tcp	25
10.1.3	Instalar djbdns	25
10.2	Uso de RPMs	26
10.3	Uso de ports en OpenBSD 2.9 y anteriores	26
10.4	Uso de Ports en FreeBSD	26
10.5	Uso de ? en Solaris	27
10.6	Guión de inicio estilo SysV	27
11	uso de los guiones add-* para la creación del fichero de datos de tinydns	29
12	enlaces útiles	29
13	Reconocimientos	29

1 Introducción

Este documento probablemente nunca sea tan útil y detallado como su inspirador, *Mi vida con qmail* de Dave Sill, pero también es cierto que djbdns no es ni de lejos tan complejo como qmail (pues realiza una tarea muchísimo más simple).

djbdns es un sencillo conjunto de programas que sirven y resuelven datos DNS. Su finalidad es servir como sustituto de BIND en muchos conceptos, aunque no incluya cada característica implementada por BIND, y quizás nunca lo haga; djbdns incluye características que puedan demostrar ser útiles, y hay algunas características especiales que BIND ofrece y que no entran en esta categoría. Igual que qmail, especialmente en los primeros días, djbdns puede requerir cierto rediseño en la implementación de nuestro sistema.

Este documento intenta dar la imagen global, que es lo que Usted necesita para ponerse en marcha; describir los componentes de djbdns y cómo encajan juntos; proporcionar cuadros de instalaciones típicas, y quizás por el camino contestar a algunas de las dudas más frecuentes. No pretende, empero, ser un sustituto de ninguno de los documentos de dudas (FAQs) que existen; éste es un documento introductorio, para leerlo del principio al final, y no una referencia.

"Mi vida con djbdns" lo empezó Bennett Todd <bet@rahul.net>, que ha escrito partes sustanciales de él. Sin su extraordinaria contribución al trabajo y sin su preparación, este documento no sería lo que es hoy.

La traducción al castellano ha corrido a cargo de J. Iván Juanes Prieto *kerberos@insflug.org*.

2 Otros recursos

Página oficial de djbdns, por Dan Bernstein

www.djbdns.org por Russel Nelson

Hay también una lista de correo, dns@list.cr.yip.to. Suscríbese enviando un mensaje sin asunto ni contenido a dns-subscribe@list.cr.yip.to, la desuscripción funciona de similar manera si envía un mensaje vacío a dns-unsubscribe@list.cr.yip.to.

3 Imagen de conjunto

El servicio de nombres de dominio (*Domain Name Service*, DNS) es una base de datos distribuida que soporta delegación de autoridad para segmentos del espacio de claves. Se usa principalmente para proporcionar una correlación entre nombres de máquina y direcciones IPv4; también ofrece correlaciones entre direcciones y nombres de máquina, información sobre máquinas, soporte especial de encaminamiento para correo, y una miríada de beneficios más exóticos y específicos de cada aplicación. Utiliza algunos tipos de registros internos para definir su propia estructura jerárquica interna, la delegación de subdominios; además, existen características para el soporte de IPv6.

Como cualquier sistema complejo que lleve correctamente a cabo un trabajo laborioso, existen una serie de conceptos con una terminología especial que se refiere a ellos. Intentaremos introducir aquí los más importantes.

El servicio DNS está compuesto de un grupo de servidores que transmiten de un lado a otro i<registros de recursos> (RRs). Hay muchos tipos de registros de recursos, y varios protocolos diferentes para solicitar dichos recursos. DNS se transmite tanto sobre TCP como sobre UDP, en el puerto 53. Las peticiones más corrientes son sobre UDP. TCP se usa cuando el total del conjunto de RRs de una réplica excede los 512 bytes, o para llevar a cabo "transferencias de zonas" (más sobre ello infra).

3.1 Resolutores y servidores de nombres: definiciones

Los autores del estándar de DNS (RFCs 1034, 1035, y muchos posteriores) lo describen basándolo en un modelo de diseño que sigue el *Berkeley Internet Name Daemon* (BIND); este modelo incluye la noción de que todos los servidores en realidad son básicamente el mismo tipo de código, e incluye la notación para ficheros de datos de zona que usa BIND. Desafortunadamente el diseño de BIND tal como lo presentan los RFCs adolece de una indeseable complejidad, y ha producido algunos contratiempos de seguridad; el diseño de `djbdns` está basado en un modelo de diseño mucho más simple, que vamos a describir. También, y por desgracia, esto ha producido un problema de terminología: existen varias funciones que puede desempeñar un servidor de DNS, y que no se distinguen claramente en la terminología DNS tradicional, porque BIND las lleva a cabo todas a la vez; según están separadas en `djbdns`, es importante darles nombres bien definidos. Llamaremos *biblioteca resolutora* a la biblioteca de rutinas que un programa cliente utiliza para realizar una búsqueda; llamaremos al demonio de apoyo al que ésta envía la petición un *resolutor recursivo*, y llamaremos a la autoridad última que son la fuente final de la información *servidores de nombres autorizados*.

Esto lamentablemente lleva a más explicaciones: pero tenga en cuenta que aunque sean más detallados, hacen dos tareas: primero, describen con precisión las tareas que se están realizando; y segundo, son razonablemente consistentes con el uso tradicional.

3.2 Tipos de consultas DNS

Pueden enviarse dos tipos de consultas DNS distintas, y que se distinguen por un bit que se llama RD (*Recursion Desired*, se solicita recursividad). Las consultas recursivas, cuando este bit está activado, las envían normalmente programas de usuario, usando rutinas en las bibliotecas del sistema, como por ejemplo `gethostbyname(3)`. Normalmente, éstas enviarán la consulta a otro servidor, cuya dirección hallarán consultando `/etc/resolv.conf` para saber qué servidor de nombres deben usar. Una consulta recursiva le pide al servidor que haga lo que sea preciso para encontrar la respuesta, incluyendo la consulta recursiva a cualesquiera otros servidores que tenga que consultar para darnos la respuesta; de ahí su nombre. La respuesta a una consulta recursiva debiera ser la respuesta final a la pregunta, o una declaración firme de que la respuesta no se pudo hallar.

El otro tipo de consulta es una consulta no recursiva, también llamada consulta iterativa; típicamente la envía un programa que actúa como resolutor recursivo; dicho programa estaría a la escucha en una dirección que el cliente puede hallar en `/etc/resolv.conf`. Así pues, un cliente que quiera encontrar la dirección del nombre de una máquina, o el nombre de la máquina a partir una dirección, o que quiera hacer cualquier otra consulta al sistema DNS, buscaría la dirección del resolutor recursivo apropiado en el `/etc/resolv.conf` local y enviaría su petición recursiva, con el bit RD activado. Esta funcionalidad se halla en `gethostbyname(3)` y `gethostbyaddr(3)`. El resolutor recursivo comienza entonces el proceso de averiguar realmente la información solicitada por parte del usuario, de una manera bastante hábil (y compleja) usando una serie de *consultas iterativas* (con el bit RD sin activar). Estas consultas iterativas no le piden al servidor que localice la respuestas en nuestro lugar mediante peticiones sucesivas, sino que simplemente le piden al servidor que nos conteste a la pregunta, o que nos diga quién sabe la respuesta mejor que él.

Así pues, allí donde una consulta recursiva para `www.example.com` podría devolver su dirección IP, un una declaración firme de que no hay tal máquina, una consulta iterativa podría devolver las identidades de otros servidores de nombres para probarlos en la búsqueda de la respuesta.

3.3 Estructura de los nombres de dominio

Los nombres de dominios son nombres para nodos de una estructura en árbol. La raíz del árbol de nombres de dominio es ".", aunque dicha raíz no se escribe normalmente en los nombres de dominio. Puede que un

nombre de dominio se escriba normalmente "www.example.com", pero la versión completa del nombre de dicho dominio es *www.example.com.* con el punto incluido.

Los usuarios normales no tienen que escribir las cosas de esa manera, y Usted no necesitará tampoco el punto final al configurar djbdns, pero puede ser útil saberlo al hacer las pruebas del DNS. Mediante el uso explícito de un "." al final, puede evitar que sus bibliotecas resolutoras intenten añadir el nombre de dominio local, según se haya configurado en la entrada clave "search" dentro de */etc/resolv.conf*. Así pues, si su dominio es *loque.sea*, y solicita *www.example.com*, quizás su navegador (dependiendo de los detalles de su biblioteca local) intente buscar *www.example.com.loque.sea* antes de intentar buscar *www.example.com* según se desea. Si en vez de eso solicita Usted *www.example.com.* el punto del final evitará que se haga aquella comprobación extra.

La estructura de árbol del espacio de nombres de DNS refleja directamente una estructura de árbol de servidores de nombre autorizados. Los servidores de nombres raíz, cuyas direcciones se configuran en cada resolutor recursivo para "encontrar una salida" y ponerlos en marcha, residen en la raíz del árbol de autoridad; los enlaces dentro del árbol siguen delegaciones de autoridad.

Estos servidores raíz pueden decir con autoridad qué servidores de nombres tienen datos para los diferentes dominios que caen directamente bajo ".", como por ejemplo "com", "net", "edu", "org", y los muchos dominios de primer nivel para países.

3.3.1 Tipos comunes de registros de recursos (RRs)

DNS soporta muchos tipos de registros de recursos. Una media docena de ellos se usan de manera general. Se describirán con más detalle infra. Pero sin 3 tipos de registro no podríamos trazar la ilustración más simple, así que incluyámoslos aquí:

A Registro de máquina, da la dirección IP de un nombre de máquina

NS registro de delegación, especifica el nombre de dominio de un servidor de nombres usado para encontrar información adicional sobre un dominio.

PTR registro de puntero pointer record, correlaciona un nombre de dominio sobre otros, y se usa generalmente para búsquedas inversas, para traducir direcciones IP otra vez a sus nombres de máquina

3.3.2 Ejemplo: encontrar la dirección IP de *www.example.com*

Cuando un cliente desea buscar la dirección de *www.example.com.*, envía al resolutor recursivo cuya dirección halla buscando en */etc/resolv.conf*. Si todo sucede correctamente, acabará (dado el caso) por encontrar una respuesta.

Dicho resolutor comenzará entonces un proceso de largo alcance para averiguar la respuesta. En la práctica, lo normal es que sea razonablemente rápido, puesto que los resolutores cachean, con gran inteligencia, la respuesta a las preguntas, y así muchas veces pueden responderlas sin tener que volver a pasar por el proceso de la búsqueda completa esbozada aquí. Pero resulta de suma importancia comprender cómo funciona dicha búsqueda, así que vamos a ello.

Un resolutor recursivo se configura con las direcciones de los servidores de nombres raíz autorizados. Empieza enviando una consulta iterativa (RD bit sin activar) preguntando por el registro A de *www.example.com* a

uno de los servidores raíz autorizados. Puede elegir uno de ellos aleatoriamente; puede haber recolectado datos sobre la rapidez con la que responden y luego intentar enviarla al que aparentemente respondía más rápido; puede que los use rotativamente. Depende del autor del resolutor recursivo.

Ahora bien, `www.example.com` probablemente no esté en los datos autorizados del servidor raíz de nombres; no sabe nada de `www.example.com`. Sin embargo, tiene una lista de servidores que son autorizados para `.com.`; de echo, conoce a cada servidor que es servidor autorizado para cada dominio de primer nivel (TLD). Ésa es precisamente la definición de un servidor raíz de nombres.

Así pues, el servidor de nombres raíz autorizado le devuelve una réplica que dice "lo siento, no tengo la respuesta para `www.example.com`, pero estos son los servidores autorizados para `.com`, y, a propósito, aquí están sus direcciones. Puesto que el resolutor recursivo tendría que preguntar de todas formas en seguida por las direcciones para obtener las respuestas, se incluyen en la respuesta del servidor raíz de nombres los registros A que correlacionan los nombres de máquina con los servidores de nombre de primer nivel para `.com`", junto con los registros NS que dan las delegaciones de `.com` a dichos servidores. A estos registros "A" adicionales se les llama "registros de unión"; recuérdelos, son importantes para comprender cómo crear mejores ficheros de datos para djbdns.

Por tanto, el resolutor recursivo ya sabe a quién plantear preguntas que terminen en `.com`, y sabe sus direcciones. Así pues, toma una (tal y como se expuso anteriormente) y envía exactamente la misma consulta que envió anteriormente: una consulta iterativa (bit RD sin activar) para el registro A de `www.example.com`. Ahora bien, el servidor de nombres TLD probablemente tampoco sepa nada de las interioridades de `example.com`; bastante ocupados están dándole a la gente información de `example.com` en su globalidad (y sobre cualquier otro dominio que termine en `.com`). Aún más importante, los detalles sobre las interioridades de `example.com` los gestionan los administradores de `example.com`, no los administradores globales de los servidores de nombres TLD. Una vez más, el resolutor recursivo no obtiene la respuesta a su pregunta, sino que obtiene el registro NS que le da los servidores de nombres autorizados para `example.com`, junto con `_sus_` registros de unión.

Como un aparte en este punto, DNS todavía funciona si faltan los registros de unión, pero es más lento debido a que los resolutores recursivos tienen que volver sobre sus pasos e ir preguntando los registros A de los servidores de nombres (y es bien fácil quedarse atrapado aquí, puesto que los servidores de nombres de `example.com` normalmente estarán `_dentro_` de `example.com`, y no podrá encontrar sus direcciones de la manera habitual). Los registros de unión son importantes.

Así pues, el resolutor recursivo sabe que los servidores de nombre autorizados para `example.com` son (se toma como ejemplo, siguiendo el uso convencional y habitual) `"a.ns.example.com"` y `"b.ns.example.com"`. Otra convención habitual es `ns1.example.com` y `ns2.example.com`. El resolutor recursivo tomará uno de ellos y enviará su pregunta sobre el registro A una vez más, y esta vez, le estará preguntando al servidor autorizado de `example.com` cuál es el registro A de `www.example.com`, y tendría que obtener una respuesta.

Luego, la respuesta vuelve al cliente. Por supuesto, cuando el navegador web sigue preguntando en seguida por la dirección de `imaginees.example.com`, para empezar a llenar los gráficos de la página, el servidor de nombres no tiene que estar volviendo a los servidores raíz, ni tiene que vérselas otra vez con el servidor TLD, sino que recuerda a quién tiene que hacerle las preguntas acerca de `example.com`, y le envía su consulta directamente.

4 Los componentes de djbdns: cómo interactúan

4.1 dnscache

El paquete djbdns nace de dos observaciones: en primer lugar, muchos problemas de seguridad con BIND surgen por la forma en la que bind toma decisiones sobre las respuestas en las que confiar, y sobre las

repuestas que descartar; los "registros de unión" funcionan porque cualquier respuesta puede contener en ella información adicional, y lo que es más, puede falsificarse cualquier respuesta. La forma más sencilla de solventar este problema era crear un nuevo resolutor recursivo que aplique reglas de seguridad estrictas en lo que respecta a la identidad del consultado, y en lo tocante las partes de la respuesta que son de confianza y las que no. Así nació dnscache, que es *_unicamente_* un resolutor recursivo; a diferencia de BIND, nunca devuelve datos autorizados, y nunca devuelve datos que no provengan directamente de un servidor de nombres autorizado, cuya autoridad haya comprobado mediante el rastreo de la cadena de delegaciones de NS desde los servidores raíz que tenga configurados. Para proporcionar flexibilidad y sencillez, puede también configurarse con servidores de nombre autorizados que sean distintos para dominios específicos. Esto hace sencillo configurar servidores de DNS "de división de horizonte" (split-horizon) que se discuten más abajo. dnscache tiene opciones configurables para controlar el tamaño del caché, y para controlar en qué interfaces aparece listada.

4.2 tinydns y tinydns-data

tinydns es probablemente el servidor de nombres autorizado totalmente funcional más pequeño posible. Sirve datos autorizados solamente una vez: cualquier consulta que no se pueda contestar desde su base de datos del disco, simplemente no se contesta. La base de datos del disco tiene estipulado decirle a *tinydns* que tiene el conocimiento completo de un dominio específico; sólo si está activado (indicado por la presencia de un registro "." en el fichero de datos), *tinydns* retornará NXDOMAIN, la constatación oficial de que el dominio solicitado no existe.

tinydns edita una base de datos muy pequeña y eficiente en formato *cdb*. Esta base de datos tiene la característica de rendimiento siguiente: cuando está en un uso activo razonable (y por tanto el bloque de encabezado al principio de la base de datos permanece registrado en la caché), cualquier consulta puede consultarse mediante dos accesos a disco. Para servidores de nombre de muy alto rendimiento, esto es inadecuado; de todas formas, el formato *cdb* no es tan abultado que no permita a servidores preparados mantener la base de datos en orden de servicio entera en la memoria, bien mediante el uso de un disco de ram explícitamente preparado, bien asegurándose simplemente de que el servidor utiliza la suficiente memoria de almacenamiento intermedio (buffer) para permitir que la totalidad de la base de datos esté en caché.

La base de datos de *tinydns* está organizada de tal manera que su clave es la pregunta, y su réplica es la respuesta; todo el conocimiento real de las estructuras de dns está en el programa (sin conexión a la red) *tinydns-data*, que traduce el fichero de texto *data* a la base de datos binaria *data.cdb*. Este programa lee un formato limpio y ligero que permite con gran sencillez codificar los registros más comúnmente utilizados; con el paso del tiempo se han ido añadiendo más tipos de registro al soporte. Para todos los registros que actualmente no estén entre los casos especiales, hay un formato de registro "genérico", y un preprocesador, que puede usarse para producir cualquier registro que se desee en dicho formato. Véase la *documentación de tinydns-data*.

4.3 axfrdns

tinydns sólo sirve datos a través de UDP, puerto 53. Hay dos casos en los que DNS precisa de servicio a través de TCP, puerto 53. Uno de ellos es cuando la réplica a una consulta excede los 512 bytes. En ese caso, la réplica queda truncada, se activa un bit que indica el truncamiento, y si el programa que está a la escucha desea la réplica completa, repite la consulta vía TCP. Ello no representa precisamente la celeridad personificada, y normalmente interesa evitarlo. Siempre que se asegure de que no codifique su fichero de datos para producir demasiada unión redundante, no tendría por qué caer en este problema a menos que haga alguna configuración realmente exótica, como una enorme granja de servidores que hagan DNS estático por *round-robin*. Observer que los ejemplos, por otra parte bien conocidos, de sitios que antes lo hacían, han ido desapareciendo; las réplicas DNS gigantes simplemente no funcionan tan bien.

El otro caso en el que se usa TCP en los DNS es en caso de *transferencia de zonas* (AXFR). Las transferencias de zona no son un mecanismo especialmente seguro para replicar los datos del DNS; se aconseja encarecidamente que utilice por ejemplo *rsync* sobre *ssh* en su lugar. Éste es el mejor modo de configurar una replicación de confianza, robusta, eficiente y segura entre servidores *tinydns*. Sin embargo, cuando *tinydns* y otros servidores de nombres (como BIND) necesitan intercambiar zonas completas, el único común denominador es AXFR. *djbdns* incluye ambas funcionalidades, la de servidor (en *axfrdns*) y cliente AXFR (*axfr-get*, véase más abajo). Se pueden usar para llevar a cabo transferencias de zona para replicación (servidores de nombres "secundarios"), así como para convetir entre el formato de datos de zona de *BIND* al formato de *tinydns-data* y viceversa.

axfrdns está diseñado para ejecutarse en paralelo con *tinydns*, en el puerto TCP 53, y sirve los mismos datos binarios obtenidos del mismo fichero de la base de datos binaria (*data.cdb*). Se ejecuta sobre el programa *tcpserver*, del paquete *ucspi-tcp* (véase infra).

Hay dos controles que permiten restringir lo que *axfrdns* autorizará o no. En primer lugar, *tcpserver* ofrece restricciones basadas en la dirección IP de origen, que se especifica utilizando *tcprules*. La acción que se tomará cuando una IP coincide puede ser denegar la conexión o permitirla; en caso de que se permita, pueden activarse ciertas variables de entorno. Esto le permite enlazar con el segundo control, la variable de entorno AXFR, una lista de dominios para los que se permite la transferencia de zonas, separados por barras inclinadas.

Supongamos que desea permitir a cualquiera que conecte con *axfrdns* para el reintento de consultas con réplicas voluminosas (mayores de 512 bytes) pero sólo desea permitir transferencias de zona desde *example.dom*.

```
=example.dom:allow :allow,AXFR=""
```

Observe que, puesto que *axfrdns* exporta los datos de *tinydns* en formato de transferencia de zona, constituye (entre otras cosas) una herramienta de conversión en formato de datos de zona: simplemente use una herramienta como *dig* para convertir la transferencia de zona al otro formato.

4.4 axfr-get

axfr-get es el cliente de transferencia de zona que viene con *djbdns*. Se ejecuta bajo *tcpclient*, y escribe un fichero de datos en formato *tinydns-data*. Puede usarse para configurar un servidor *tinydns* como secundario de un servidor *BIND*. También puede usarse (con la publicación *axfrdns*) para relaciones maestro-esclavo entre servidores *tinydns*, pero no se recomienda para dicho uso; *rsync* sobre *ssh* es preferible de todo punto, pues le supera en simplicidad, rendimiento y seguridad. El formato de salida escrito por *tinydns* tiene como objetivo primero (y primario) asegurar que se conserva cualquier formato de registro posible; retorno al formato de registro genérico cada vez que lo necesita. Crear datos que sean agradables y adecuados de mantener a mano es un objetivo secundario en este programa. Hay algunos guiones en la *página de djbdns* que pueden ayudar a limpiar los datos para hacerlos más agradables de mantener a mano.

4.5 El programa cliente de DNS

El paquete *djbdns* incluye algunas utilidades para llevar a cabo algunas de las tareas para las que se usan **host(1)**, **dig(1)** y **nslookup(8)**.

En las sinopsis, *FQDN* significa que debe especificarse un nombre de dominio totalmente calificado, que es diferente del nombre de máquina (es decir, el nombre especificado no tendrá anexo el nombre de dominio local). La notación es

```

nombre_orden argumentos -> salida línea generada
otros posibles comentarios

```

Estas son las utilidades para guiones; útiles para pruebas o verificaciones, pero pensadas como bloques de construcción de otros programas mayores; su salida se limita al tipo de datos desnudos válidos para depuración, y se halla reducido para hacerlo lo más sencillo posible y que sirva de entrada a otros programas.

```

dnsip FQDN -> dirección_ip
dnsipq nombre_máquina
-> FQDN dirección_ip
dnsname dirección_ip
-> FQDN
dnsmx FQDN
-> preferencia FQDN
Si no existe registro MX, dnsmx lo "falsifica" con una
preferencia de zero (incluso si no hay registros de ningún
tipo, dnsmx no-hay-máquina devolverá "0 no-hay-máquina".
dnstxt FQDN
-> registro text si hay alguno asociado con el FQDN
línea en blanco si no se halló ninguno

```

Hay más herramientas de puro diagnóstico; su salida está menos dirigida al uso de otros programas y más al análisis humano, en la depuración. En estas órdenes, "tipo" es en tipo de consulta DNS, como por ejemplo "ptr", "a", "txt", "mx", "ns", "soa", or "any".

```

dnsq tipo FQDN servidor -> salida de consulta para depurado
Envía una solicitud iterativa al servidor indicado; es ésta
la herramienta preferible para comprobar tinydns.
dnsqr tipo FQDN
-> salida de consulta para depurado
Esta orden envía una solicitud recursiva, adecuada para
comprobar dnscache. Si necesita ignorar el cache por
defecto especificado en /etc/resolv.conf, puede configurar
la variable de entorno DNSCACHEIP:
DNSCACHEIP=x.y.z.t dnsqr tipo FQDN
dnstrace tipo FQDN servidor
-> rastro de depuración de una búsqueda completa
El servidor especificado aquí debe se un servidor de
nombres _raíz_; dnstrace lanzará una búsqueda del FQDN
comenzando desde ese root, y mostrará todos los posibles
caminos para responder a la pregunta; ello normalmente
revelará un host maligno.

```

4.6 dnsfilter

Este programa está diseñado para la conversión masiva de elementos como por ejemplo los ficheros de registro o bitácora; también lleva a cabo una terrorífica prueba que pone al límite a un resolutor recursivo. Lee líneas desde entrada estándar y las escribe por salida estándar. Si el primer campo separado por espacios de una

línea se asemeja a una dirección IP, dnsfilter intenta hacerle una búsqueda inversa (dada una IP x.y.z.t, intentará una solicitud PTR para t.z.y.x.in-addr.arpa.) Si tiene éxito, la información de dominio se anexa a la dirección IP en la línea de salida. Estas consultas se realizan en paralelo; las opciones de línea de órdenes pueden permitirle controlar los detalles. Véase *la documentación* para más detalles.

4.7 Servidores especializados

En principio las tareas que estos servidores llevan a cabo podría realizarlas *tinydns*, pero son lo bastante específicas como para precisar un código específico que realice mejor la tarea.

`pickdns` Desde 1.04, `pickdns` ya no es preciso. Su funcionalidad es ya parte de `tinydns`.

Servidor autorizado de balanceo de carga, diseñado para usar DNS distribuyendo la carga de los clientes en múltiples servidores; incluye facilidades para redigirir clases concretas de clientes a subconjuntos específicos de una granja de servidores distribuidos (por ejemplo, para intentar redigirir a los usuarios a servidores más cercanos a ellos).

`walldns`

sirve datos DNS autorizados puramente genéricos, relacionados con todo lo de un área concreta: para todas las direcciones IP x.y.z.t, actúa como (en la notación `tinydns-data`):

```
=t.z.y.x.in-addr.arpa:x.y.z.t
```

`rbldns`

Para servir datos del tipo RBL; responde a consultas A, TXT, o * para t.z.y.x.\$BASE si x.y.z.t está listado en su fichero de datos, que puede especificar direcciones y subredes individuales, así como las réplicas que haya que enviar.

4.8 daemontools

daemontools es un paquete complementario, necesario y previo a *djbdns*. Proporciona algunos programas de ayuda que nos asisten en la tarea de lanzar y administrar demonios.

`svscan` hace una tarea vagamente similar al `init` del System V: monitoriza los trabajos y se asegura de que se mantienen

en ejecución. Normalmente comienza con el directorio `/service`, y cada subdirectorio de dicho directorio define un servicio; normalmente se trata de enlaces simbólicos a directorios dentro de `/etc`, que a su vez se crean inicialmente gracias a determinados programas `-conf` (`dnscache-conf`, `tinydns-conf`, etc).

`svc` proporciona una interfaz interactiva al usuario para controlar `svscan`; toma como argumento una opción como `"-u"` para "up" (levantar), `"-d"` para down (echar abajo) o `"-t"` para "reiniciar", seguido del nombre del servicio en la forma del camino hasta el directorio bajo control.

El caso más común es
`svc -t /service/dnscache`
 para reiniciar el dnscache, que es la forma más sencilla de forzarlo a que refresque los datos cuando lo que hay el en caché no es de su agrado.
 supervise lo ejecuta `svscan` para vigilar un demonio específico, y lo reinicia si muere.
`multilog` lee datos de registro de bitácora (logs) desde la entrada estándar, los filtra si se desea y deposita los resultados en uno o más ficheros de registro (log), encargándose automáticamente de su rotación.

4.9 ucspi-tcp

UCSPI es el *UNIX Client-Server Program Interface* (interfaz de programa cliente-servidor para UNIX). Define una estructura de línea de órdenes y unas especificaciones de variables para programas de ayuda a las comunicaciones entre procesos; ello hace más sencillo escribir clientes y servidores.

UCSPI-TCP es la variedad específica de UCSPI para aplicaciones TCP; especifica detalles adicionales sobre las variables de entorno específicas y otros detalles de esta índole.

ucspi-tcp es el paquete en el que *djb* implementa UCSPI-TCP.

`tcpserver` como `inetd`, pero para un único servicio. Una invocación de `tcpserver` se pondrá a la escucha de conexiones en un puerto dado; cuando tenga lugar una, pondrá en marcha un programa cliente con los argumentos especificados en la línea de órdenes de `tcpserver`, y con sus variables de entorno según las especifique UCSPI-TCP. `tcpserver` implementa reglas de control de acceso.
`tcprules`
 compila en una base de datos `cbd` las reglas de control de acceso para `tcpserver`
`tcpclient`
 un programa de asistencia para clientes; realiza la configuración para escribir cliente de red para protocolos TCP, siguiendo las especificaciones UCSPI-TCP.

5 Configuraciones de ejemplo

Sigue un surtido de ejemplos ilustrativos. Están tomados de experiencias concretas en el trabajo, y se orientan más a reflejar las circunstancias que varios usuarios han encontrado comunes y rutinarias, antes que a ilustrar principios o técnicas específicos de una forma precisa.

5.1 Configuración sencilla

Ya está bien de teoría; comencemos. No se explicará aquí lo de la descarga, compilación, instalación y adición de usuarios; puede encontrar dicha información en el Apéndice junto con notas específicas de cada sistema operativo. También se mencionan varios métodos de instalación alternativos. En primer lugar, ha

de decidir qué paquetes necesita. Normalmente necesita que *dns-cache* sea su resolutor y que *tinydns* sea su servidor de nombres autorizado. Si tiene que mantener servidores secundarios que ejecuten *BIND*, también necesitaría *axfrdns* (esto no es 100% cierto; si hace algunos trucos extraños y las respuestas simples crecen hasta más allá de los 512 bytes, entonces también necesitará *axfrdns*, pero no es lo más normal). Tras la instalación, ha de configurar y poner en marcha los servicios, según se describe más abajo. Para ello hay que usar *daemontools* (y *ucspi-tp* si está usando *axfrdns*) en las ubicaciones por defecto.

5.1.1 Configuración de *dns-cache*

Necesitará crear dos usuarios, llamados *dns-cache* y *dnslog*. Por razones de seguridad, asegúrese de que no pueden hacer login al sistema.

Ejecute *dns-cache-conf*:

```
dns-cache-conf dns-cache dnslog /etc/dns-cache 1.2.3.4
```

donde 1.2.3.4 es la IP en la que *dns-cache* ha de escuchar. Por defecto, los ficheros de bitácora (logs) residen en */etc/dns-cache/log/main*. Si desea que sus ficheros de bitácora estén en */var/log/dns-cache*, cree dicho directorio con propietario *dnslog* y sustituya *./main* en */etc/dns-cache/log/run* por */var/log/dns-cache*.

Inicie *dns-cache* informándole a *svscan* acerca de él:

```
ln -s /etc/dns-cache /service
```

dns-cache debería arrancar antes de 5 segundos.

Si *dns-cache* va a ser solamente su resolutor local, entonces ya ha terminado. Por defecto, *dns-cache* no acepta consultas desde otras máquinas. Si quiere que acepte consultas desde 1.2.3.*, simplemente haga

```
touch /etc/dns-cache/root/ip/1.2.3
```

Puede añadir o eliminar redes sin tener que informar de ello al *dns-cache* que está en ejecución.

5.1.2 Configuración de *tinydns*

De nuevo son necesarios dos usuarios, *tinydns* y *dnslog*. Si ha configurado previamente *dns-cache* según se describe más arriba, *dnslog* ya existirá.

Ejecute *tinydns-conf*:

```
tinydns-conf tinydns dnslog /etc/tinydns 1.2.3.5
```

donde 1.2.3.5 es la IP en la que *tinydns* debe escuchar. Observe que **no puede usar la misma IP que ha usado para *dns-cache***. Por defecto, los ficheros de bitácora habitan en */etc/tinydns/log/main*. No nos agrada tener nada distinto a ficheros de configuración en */etc*; si desea tener sus propios ficheros de bitácora en */var/log/tinydns*, cree este directorio con propietario *dnslog* y sustituya *./main* en */etc/tinydns/log/run* por */var/log/tinydns*.

Arranque *tinydns* informándole a *svscan* de su existencia:

```
ln -s /etc/tinydns /service
```

tinydns ha de arrancar antes de 5 segundos.

Debes decirle a *tinydns* los servidores que ha de resolver. Abra */etc/tinydns/root/data* en su editor favorito y escriba:

```
#define el servidor de nombres autorizado .example.com::ns1.example.com
#intercambiador de correo
@example.com::mail.example.com
#IP para maquina1,2,3,4,5
=maquina1.example.com:1.2.3.1
=maquina2.example.com:1.2.3.2
=maquina3.example.com:1.2.3.3
=maquina4.example.com:1.2.3.4
=maquina5.example.com:1.2.3.5
#maquina5 tiene también el nombre ns1
+ns1.example.com:1.2.3.5
#maquina1 es nuestro servidor de correo
+mail.example.com:1.2.3.1
#maquina1 es también nuestro servidor web
+www.example.com:1.2.3.1
```

Tras editarlo, cámbiese al directorio `/etc/tinydns/root` y ejecute "make". Ello compilará el fichero `data.cdb` que lee `tinydns`.

También hay guiones en `/etc/tinydns/root` que pueden evitarle la tarea de editar el fichero de datos. No los juzgamos demasiado útiles puesto que sólo le permiten añadir elementos, no eliminarlos o editarlos. Véase el apéndice para ejemplos sobre cómo usarlos.

5.1.3 Configuración de axfrdns

Mismo procedimiento que el anterior; cree el usuario axfrdns y ejecute:

```
axfrdns-conf axfrdns dnslog /etc/axfrdns /etc/tinydns 1.2.3.5
```

donde 1.2.3.5 es la IP en la que axfrdns debe estar a la escucha. Normalmente debe ser la misma IP en la que `tinydns` está a la escucha. Debe definir qué servidores (direcciones IP) le pueden pedir una transferencia de zona. Si desea permitir a 9.8.7.6 transferencias de zona para `example.com` y `example2.com`, añada esta línea a `/etc/axfrdns/tcp`:

```
9.8.7.6:allow,AXFR="example.com/example2.com"
```

5.2 split horizon

5.3 DNS secundarios

5.3.1 tinydns a tinydns

copie el fichero data.cdb Puede simplemente copiar los datos generados `data.cdb` a la segunda máquina (usando `rsync` sobre `ssh`, por ejemplo). El fichero `data.cdb` resulta ligeramente más grande que el fichero `data`, así que esta opción sólo es útil en redes locales. Nota: `data.cdb` es independiente de la arquitectura, así que no hay problema incluso si usa un sistema operativo distinto. No es nuestro método predilecto. Almacenar los datos en varias máquinas me permite una medida extra de seguridad si la primaria falla. Su caso puede ser distinto.

Si copia `data.cdb`, no necesita hacer nada en la parte del secundario para actualizar los nuevos datos, ya que `tinydns` trabaja directamente con el fichero de disco y se dará cuenta de la actualización inmediatamente.

Sin embargo debe asegurarse de que el fichero no se mueve a su destino hasta que no haya terminado completamente la copia, así que o bien propáguelo con **rsync**, que se asegura adecuadamente de este extremo, o bien confecciónese un guión del tipo

```
scp data.cdb secondary:/etc/tinydns/root/data.cdb-new && \ ssh secondary mv
/etc/tinydns/root/data.cdb-new /etc/tinydns/root/data
```

Uso de rsync con ssh Es este el método que preferimos. Se puede usar incluso si ambos *tinydns* están conectados a través de una línea lenta, por ejemplo en localizaciones distintas. Ello es así porque sólo se transfieren los cambios, y además se utiliza compresión de los datos.

```
rsync -e ssh -az /etc/tinydns/root/data $host:/etc/tinydns/root/data ssh $host "cd
/etc/tinydns/root; make"
```

donde \$host es su servidor *tinydns* secundario. NB: tanto rsync como ssh tienen un problema común, a saber, que otros programas quieren ejecutarlos en varios contextos; por ejemplo, rsync quiere ejecutar ssh, y rsync quiere invocarse a sí mismo mediante ssh en el sistema remoto. Es más, ambos paquetes se instalan por defecto en */usr/local* cuando se instalan desde código fuente con la configuración por defecto, y */usr/local/bin* no está en el PATH por defecto en la mayoría de los intérpretes de órdenes. El resumen de lo anterior es que quizás necesite tomar la orden anterior y reescribirla de forma semejante a

```
/usr/local/bin/rsync -e /usr/local/bin/ssh \ -rsync-path=/usr/local/bin/rsync ...
en lugar de la más sencilla
rsync -e ssh ...
```

Por esta razón, generalmente hacemos la prueba y nos aseguramos de que los caminos */usr/bin/rsync* y */usr/bin/ssh* funcionan incluso si no son los caminos por defecto, y añadimos enlaces simbólicos en nuestros sistemas para hacerlos funcionar en caso necesario.

5.3.2 Transferencia de zonas de BIND a tinydns

(sería interesante hacer que *tinydns* reaccione a notificaciones entrantes de estilo BIND, lanzando un guión externo que haga la comprobación de autenticación y que llame a axfr-get: ¿algún voluntario para escribir el parche?)

(ACTUALIZACIÓN: Tenemos el parche, pero sin tiempo para haberlo revisado aún).

5.3.3 Transferencia de zonas de tinydns a BIND

Si el DNS secundario no ejecuta *tinydns*, y se encuentra bajo su control, debiera actualizarlo a *tinydns*. Si no está bajo su control, considere el uso de otro secundario ;-)). Bien, basta de bromas: si realmente tiene que dar soporte a un secundario que ejecuta otro servidor DNS, tiene que implementar notificaciones de estilo BIND. Estas notificaciones dicen a BIND "el dominio xyz ha cambiado, renuévalo). Así BIND comprueba si el número de serie en su *tinydns* es mayor que el que tenía, y si lo es, BIND lanzará un AXFR para este dominio (que quiere decir que obtiene los datos y los recarga). El Notify tiene que tener la IP de su *tinydns* como origen, y ello hace la tarea un poco más complicada, si la IP de *tinydns* no es la misma que la IP primaria de su máquina. Observer que previamente debe haber configurado axfrdns, para que este proceso funcione.

El guión Notify en sí (desarrollado por Jos Backus) no es demasiado complicado:

```
#!/usr/bin/perl -w
# uso: dnsnotify zona esclavo ...]
#ejemplo : dnsnotifyexample.org1.2.3.41.2.3.5

useNet :: DNS;
useData :: Dumper;
usestrict;

my$usage = "uso : dnsnotifyzonaesclavo...]\n";

my $zone = shift;
my @slaves = @ARGV;

die $usage unless defined($zone) && @slaves;

my $type = "SOA";
my $class = "IN";

my $packet = new Net::DNS::Packet($zone, $type, $class);
die unless defined $packet;

$packet->header->opcode("NS NOTIFY OP");
$packet->header->aa(1);
$packet->header->rd(0);

#$packet->print;

my $res = new Net::DNS::Resolver;
$res->LocalAddr("1.2.3.4");
my $reply;

for (@slaves) {
    $res->nameservers($);
    # print Dumper($packet);
    $reply = $res->send($packet);
    if (defined $reply) {
        # $reply->print;
    } else {
        print "\n; TIMED OUT\n";
    }
}

exit 0;
```

Hay un cambio con respecto a la versión publicada por Jos: La línea `$res->LocalAddr`, que establece la IP origen de las consultas de manera que el BIND del otro extremo acepte el Notify. Por desgracia tendrá que parchear `Net::DNS::Resolver` para que soporte esta característica. El parche (desarrollado por `jeff@lfcity.org`) tampoco es muy complicado. Obténgalo de <http://www.lifewithdjb.org/Resolver.pm.patch>. El desarrollador de `Net::DNS` está informado; es de esperar que

`:DNS` soporte `LocalAddr` pronto sin necesidad de parchear.

5.4 Sitios grandes con necesidad de ficheros separados para cada zona

5.5 Uso de djbdns en entornos de ISP

5.5.1 Exportar tinydns-data desde una base de datos relacional (RDBMS)

Existe una diferencia fundamental entre djbdns en entornos reducidos, y djbdns en un ISP: la automatización. En tanto que ISP, es probable que Usted arme sus datos DNS tomándolos de una base de datos. De esta manera, su base de datos se comunicará directamente con *tinydns* en lugar de tener que lidiar con add-ns, add-mx y similares.

El formato de este fichero de datos se documenta en <http://cr.yp.to/djbdns/tinydns-data.html>. Parece un tanto extraño al principio, debido a que no está optimizado para su lectura por humanos, sino para su análisis sintáctico. Ello hace que nuestro trabajo de "exportar desde base de datos" sea realmente sencillo.

Visualice mentalmente una serie de tablas pequeñas y sencillas, una para los datos del dominio y otro para los datos de dns en sí. Llamémoslas "Domains" y "NSEntry".

Domains podrá tener un aspecto similar a:

```
Domain varchar Domainname example.com nserver1 varchar 1st Nameserver ns1.isp.com
nserver2 varchar 2nd Nameserver ns2.isp.com
nserver3 varchar 3rd Nameserver ns3.isp.com
nserver4 varchar 1st Nameserver ns4.isp.com
nserver5 varchar 2nd Nameserver ns5.isp.com
nserver6 varchar 3rd Nameserver ns6.isp.com
mx1 varchar 1st Mailserver mx1.isp.com
mx2 varchar 2nd Mailserver mx2.isp.com
mx3 varchar 3rd Mailserver mx3.isp.com
qntp1 varchar 1st qntp-Mailserver qntp1.isp.com
qntp2 varchar 2nd qntp-Mailserver qntp2.isp.com
serial varchar Serial No. 972244824 (set to time() on every change)
SOAmail varchar Mail-addr. SOA hostmaster.isp.com (replace @ by . !!!)
NS bit 1=we do dns, 0=no 1
changed bit 1=changes were made 1 (reset to 0 after building data)
```

y NSEntry será bien sencillo:

```
Domain varchar Domainname example.com Host varchar Hostname www
Type varchar Entry-Type A
Value varchar Entry-Value 1.2.3.4
```

así pues, veamos cómo obtener los datos de una base de datos para *tinydns* (con perl):

```
#!/usr/bin/perl
# Obtiene de la base de datos información de DNS-Info y crea
# el fichero de datos de tinydns
# escrito por Henning Brauer, Hostmaster BSWS, julio de 2000
# Licencia: BSD
# actualización 8 de febrero de 2001

use DBI;
```

```

# conectarse a su base de datos
$dbh=DBI->connect("DBI:mysql:mydb:myhost", "login", "password") || die "No se puede
conectar con el servidor de bases de datos DBI::errstr,\n";

# buscar los cambios
$sth=$dbh->prepare("SELECT serial FROM Domains WHERE changed=1");
$rv=$sth->execute;
$sth->finish;

if ( $rv > 0 ) {
print "Reescribimos Nameserverdata\n";

#Estos valores los aceptan DENIC y CORE
$refresh=10000;
$retry=3600;
$expire=604800;
$min=86400;

$i=0; $j=0;

# Sacar copia del fichero viejo y usar la plantilla para el nuevo
system("mv /etc/tinydns/root/data /etc/tinydns/root/data.old");
system("cp /etc/tinydns/root/data.tl /etc/tinydns/root/data");
open OF, ">> /etc/tinydns/root/data";

# obtener domain-data
$sth=$dbh->prepare("SELECT Domain, nserver1, nserver2, nserver3, nserver4, nserver5,
nserver6, mx1, mx2, mx3, serial, SOAmail, NS, changed, qmtp1, qmtp2, mail FROM Domains
WHERE NS=1 ORDER BY Domain");
$sth->execute;
while ( ( $zone, $ns1, $ns2, $ns3, $ns4, $ns5, $ns6, $mx1, $mx2, $mx3, $serial, $mail,
$ns, $changed, $qmtp1, $qmtp2, $mtype ) = $sth->fetchrowarray ) {
#soporte para entradas de registros de unión (como ns.bsws.de 213.128.133.188)
foreach $dns (@ns) {
$dns=~s/^(?+).*/$1/;}

#SOA
print OF "Z$zone\:$ns1]\:$mail\:$serial\:$refresh\:$retry\:$expire\:$min\:\:\n";
#Nameserver
foreach $dns (@ns) {
if ( $dns ne "" ) { print OF "."; print OF "$zone\:\:$dns\:\:\n"; }
}

#MX - ahora con soporte MXPS (http://cr.yip.to/proto/mxps.txt)
if ( $mx1 ne "" ) { print OF "\@$zone\:\:$mx1\:12816\:\:\n"; }
if ( $mx2 ne "" ) { print OF "\@$zone\:\:$mx2\:12832\:\:\n"; }
if ( $mx3 ne "" ) { print OF "\@$zone\:\:$mx3\:12848\:\:\n"; }
if ( $qmtp1 ne "" ) { print OF "\@$zone\:\:$qmtp1\:12801\:\:\n"; }
if ( $qmtp2 ne "" ) { print OF "\@$zone\:\:$qmtp2\:12817\:\:\n"; }

# obtiene las entradas propiamente dichas
$st2=$dbh->prepare("SELECT Host, Type, Value FROM NSentry WHERE Domain=\"$zone\" ORDER
BY host");
$st2->execute;

```

```

while ( ( $host, $typ, $value ) = $st2->fetchrowarray ) {
#A-Record - Name to IP
if ( $typ eq "A" ) {
print OF "+$host\.$zone\:$value\:\:\n";
}
#A-Record + PTR for reverse lookup
if ( $typ eq "AR" ) {
print OF "=$host\.$zone\:$value\:\:\n";
}
#CNAME - bad thing...
if ( $typ eq "CNAME" ) {
print OF "\C$host\.$zone\:$value\:\:\n";
}
}
$st2->finish;

# recuerda los cambios y los servidores de nombres a los que enviar notificaciones
if ( $changed eq 1 ) {
foreach $dns (@ns) {
# sustituye bsws.de por el dominio de su dns (you your nameserverdomain)
if ( $dns !~ /\.bsws.de$/ && $dns ne "" ) {
$chng$j] = $zone;
$ntfy$j++] = $dns;
}
}
}
}
$sth->finish;
close OF;
chdir("/etc/tinydns/root");
system("make");

#note 1

$dbh->do("UPDATE Domains SET changed=0 WHERE NOT(changed=0)");
}

$dbh->disconnect();

```

Así que ya está. Si configuró *tinydns* y *axfdns* tal como se describía anteriormente, ahora dispone de un servidor de nombres que ejecuta el excelente software de djb y que obtiene sus datos de una base de datos. Llegados a este punto, es sencillo añadir una tarea de cron que ejecute el proceso anterior a intervalos regulares. También es sencillo construir algún tipo de iterfaz que permita a sus usuarios introducir ellos mismos sus datos de DNS.

5.5.2 Ejecución de servidores *tinydns* adicionales

Para la mayoría de dominios TLDs (top level domains) debe ejecutar al menos dos servidores DNS. Ejecutar un segundo *tinydns* no presenta problemas; tendrá entonces dos primarios, y no el clásico esquema primario-secundario. Esto es bueno, en caso de que su primario falle. Nos permitimos sugerirle que use `rsync` para transferir los datos al servidor *tinydns* secundario; simplemente añada las siguientes líneas al guión anterior, a la altura de '#note 1':

```
system("/usr/local/bin/rsync -e /usr/bin/ssh -az /etc/tinydns/root/data
$host:/etc/tinydns/root/data"); system("ssh $host \"cd /etc/tinydns/root; make\"");
```

donde `$host` es su segundo servidor *tinydns*. Por supuesto, también puede simplemente ejecutar el guión completo en su segundo servidor *tinydns*.

Si necesita dar soporte a DNS secundarios que ejecuten BIND, utilice el guión `dnsnotify` que se presentó anteriormente y llámelo cuando cambien los datos. Para ello, simplemente inserte la línea

```
for ( $i=0; $i<$j; $i++ ) { print "Notify: $chng$i->$ntfy$i\n"; system("/path/to/dns
notify.pl $chng$i]$ntfy$i]"); }
```

en el guión `db2tinydns` a la altura de '#note 1'.

5.5.3 Conclusión

Dispone Usted de al menos un servidor *tinydns* que obtiene sus datos de una base de datos `mysql`, y que notifica sus datos a servidores BIND secundarios o sincroniza sus datos con otros servidores *tinydns*, y todo se lleva a cabo automáticamente. Así que eso le deja tiempo de desarrollar una preciosa interfaz web para su base de datos ;-))

El guión completo está disponible en todo momento en `lwd.bsws.de`.

5.6 Servidor raíz de nombres privado

Si ha aislado redes (es decir, tiene redes sin conexión a Internet) y tiene ejecutándose servicios que requieren del funcionamiento de un DNS, tendrá que simular la infraestructura que se encuentra en Internet. La cuestión principal es que tiene que proporcionar los "root servers" que son punto de partida para todo proceso de resolución DNS. Al menos tendrá que proporcionar un servidor que sea autorizado para la raíz del espacio de nombres DNS. Afortunadamente esto es sencillo con *tinydns*. Simplemente defina los servidores de nombres para "." en su fichero de datos

```
./add-ns . 1.2.3.1
```

Puede especificar más servidores en caso de que ejecute múltiples servidores raíz privados:

```
./add-ns . 1.2.3.1 ./add-ns . 1.2.3.2
./add-ns . 1.2.3.3
```

Para todas las subzonas que defina, no es preciso que especifique servidores de nombres, si las zonas las sirve el el mismo conjunto de servidores de nombres. También puede delegar subzonas a otros servidores de

nombres si se halla en una red de una organización extensa que replica la naturaleza distribuida del DNS. Supongamos que internamente usa el dominio "local" y que el departamento de ventas ejecuta su propio servidor de nombres en 1.3.1.1.

Además define su concentrador central de correo y el servidor de la intranet. Entonces la configuración anterior cambia a:

```
./add-ns . 1.2.3.1 ./add-ns . 1.2.3.2
./add-ns . 1.2.3.3
./add-childns ventas.local 1.3.1.1
./add-mx local 1.2.3.5
./add-host intra.local 1.2.3.4
```

Para activar la resolución mediante los servidores raíz privados, ha de informar a dnscache de su existencia. Ello se consigue sustituyendo el contenido de root/servers/@ en su directorio dnscache y */etc/dnsroots.global* con las direcciones IP de sus servidores raíz privados, a razón de un por línea.

5.7 Servidores raíz alternativos

En un momento dado querrá poder resolver nombres de dominio de primer nivel que no hayan sido confirmados por ICANN. Ello no constituye problema; simplemente establezca una nueva lista de servidores raíz. Eche un vistazo a *technical hints* en *The Open Root Server Confederation*

5.8 Servir los mismos datos a múltiples interfaces

Existe un parche que permite vincular *tinydns* a múltiples interfaces. También se recomienda configurar la variable de entorno "IP" de *tinydns* con el valor 0.0.0.0. Aunque estos sistemas pueden funcionarle, no se recomiendan.

El procedimiento recomendado es ejecutar múltiples instancias de *tinydns*, una por cada interfaz que haya de servirse. La variable de entorno "ROOT" de todas las instancias se configura entonces con el mismo valor.

¿Suenan a derroche? Piénselo dos veces. El fichero *data.cdb* es memoria de solo lectura, que *tinydns* hace coincidir para que el sistema operativo comparta todos los almacenamientos intermedios entre los procesos *tinydns*. También el programa en sí puede compartirse (y se compartirá) por parte del sistema operativo, de tal manera que la sobrecarga o derroche se reduce a algunos procesos más (*supervise/tinydns*) y la memoria asociada del entorno de cada uno de ellos, que apenas representa nada dada la cantidad de memoria actual de las máquinas.

La ventaja de este proceder es que no tendrá que vincular a todas las interfaces de la máquina, sino sólo a las que Usted especifique. También obtiene la posibilidad de controlar separadamente los servidores DNS en los diferentes interfaces.

He aquí un ejemplo de la configuración para las tres interfaces:

```
tinydns-conf tinydns dnsmlog /etc/tinydns1 1.2.1.1 tinydns-conf tinydns dnsmlog
/etc/tinydns2 1.2.2.1
tinydns-conf tinydns dnsmlog /etc/tinydns3 1.2.3.1
echo "/etc/tinydns1/root" > /etc/tinydns2/env/ROOT
echo "/etc/tinydns1/root" > /etc/tinydns3/env/ROOT
ln -s /etc/tinydns1 - 3]/service
```

6 Mantenimiento

6.1 tinydns

El mantenimiento de *tinydns* se reduce normalmente a las actualizaciones, más o menos frecuentes, de su fichero de datos. Puede que tenga que optimizar su actualización para períodos largos, por ejemplo si utiliza la característica "location".

Para cambios en registros, *tinydns* proporciona una característica que permite transiciones suaves sin intervención manual: marcas temporales. Eche un vistazo a la documentación de *tinydns-data* para ver cómo la combinación de *ttl/timestamp* (tiempo de vida/marca temporal) puede usarse con este propósito. Hay un problema con el formato de las marcas temporales: nadie puede introducirlas sin calcularlas. Existe una utilidad que traduce marcas temporales en formato ISO 8601 al formato externo TAI64 requerido. Debe enlazarse contra la biblioteca *libtai* de Dan Bernstein:

```
* // -----
// isotai64.c
// Frank Tegtmeier <fte@pobox.com>, 2000-02-25
// véase http://www.lightwerk.com/djbdns/isotai64.html
//
// formato de entrada: AAAA-MM-DD HH:MM:SS zzzzz
// zzzzz es el huso o zona horaria (+0200, +0000, -0400, ...)
//
// -----
/

#include <stdio.h>
#include <time.h>
#include "tai.h"
#include "leapsecs.h"
#include "caltime.h"

char line[100];

char xTAIPACK[];

main()
{
    struct tai t;
    struct caltime ct;
    int i;

    if (leapsecs init() == -1)
        printf("unable to init leapsecs\n");

    while (fgets(line, sizeof line, stdin))
    {
        if (!caltimescan(line, &ct))
            printf("unable to parse\n");
        else
        {
            caltime tai(&ct, &t);
            taipack(x, &t);
            for (i = 0; i < TAIPACK; ++i)
                printf("%2.2x", (unsigned long) (unsigned char) x[i]);
        }
    }
}
```

```
}  
printf("\n");  
}  
exit(0);  
}
```

6.2 dnscache

Para sacarle el máximo partido a su dnscache, debe comprobar regularmente si su tamaño es el apropiado para su organización. Si su caché es demasiado grande, posiblemente no constituya mayor problema, si dispone de la memoria suficiente para dedicarla a esta tarea. Sin embargo, si su caché es demasiado pequeña, las cosas empeorarán más de lo deseable. En primer lugar, la máquina ejecutante de dnscache estará más cargada, ya que dnscache tiene que solicitar registros con mayor frecuencia. Esto vale también para servidores de contenido a los que su dnscaché hará las consultas. Una caché demasiado pequeña también incrementa el uso de ancho de banda en su conexión a Internet. Ello quiere decir también que sus usuarios habrán de esperar más tiempo por los resultados de sus consultas. En la FAQ o preguntas de uso frecuente (PUF) se describe cómo ajustar la caché. Véase también la sección "uso de memoria de dnscache" en este documento.

7 Comparación con BIND; cuestiones relacionadas con la migración

7.1 Migración

En breve:

```
-instale djbdns -ejecute axfr-get contra su BIND  
-detenga bind  
-arranque tinydns  
-vaya a tomarse un vinito de Rioja
```

7.1.1 Conversión de sus ficheros de zona

7.1.2 Si es Usted servidor primario de un servidor BIND

Ajuste de los serials Si no quiere usar registros *Z* en su configuración *tinydns*, porque es mucho más fácil usar los predeterminados de *tinydns-data*, debe echarle un vistazo a sus números de serie antes de migrar a *tinydns*. Los servidores de nombre secundarios que ejecutan BIND (y probablemente otros también) se basan en el mecanismo de AXFR para mantener su contenido en sincronía con su primario. La decisión de actualizar o no los datos recae sobre el llamado "número de serie" (serial number) incluido en el registro SOA. Las transferencias de zona sólo se llevan a cabo si el número de serie del servidor primario es *mayor* que el número de serie del servidor secundario. Puesto que los valores del número de serie están representados por valores enteros positivos de 32 bits, su rango de valores es de 0 a $2^{32}-1$ (4294967295). Para evitar el problema de alcanzar el límite superior, la comparación de números serie de específica dentro de la "aritmética de espacio de secuencias" que se explica en el *RFC1982*.

tinydns utiliza el tiempo de modificación del fichero de datos y establece el número serie en el número decimal de segundos desde el segundo cero Unix. Ello devolvía valores con 9 dígitos hasta hace pocas fechas.

Uno de los formatos comunes para los números de serie en los sistemas BIND es usar los 4 dígitos del año, los 2 del mes, los 2 del día y uno o dos para un número de incremento desde el último cambio de zona. Para el quinto cambio del día 2001-02-14 obtendríamos "200102144" o "2001021404". El formato de nueve dígitos no es problema porque el número de serie que tinydns-data crea es mayor que el generado con este formato. Por tanto, las transferencias de zona continuarán funcionando tras el cambio a tinydns.

Más problemático es el formato de 10 dígitos. Los números de serie con este formato son mayores que los que usa tinydns. Para permitir que las transferencias de zona continúen tras el cambio a tinydns, tendrá que ajustar los números de serie después del cambio.

Resumiendo, la comparación entre números de serie que se define en RFC1982 utiliza el "ángulo" más pequeño entre dos valores, si se considera el rango de valores como un anillo de valores donde 0 y 4294967295 son vecinos o colindantes. Para hacerlo más sencillo, eche un vistazo a un reloj: el cómputo de horas de un reloj nos da un rango de valores de 0 a 11 donde el número mayor (11) es colindante con el 0. Y es un anillo: todo el mundo sabe que 1(pm) es más tarde que 11(am). Así pues, siguiendo RFC1982 las siguientes comparaciones dan estos resultados:

```
5 > 2 2 > 9
4 > 11
2 and 8? -> undefined
0 and 6? -> undefined
```

Quizá desee echarle un vistazo a *esta ilustración*. para comprenderlo mejor.

Entonces, ¿cómo ajustar el número de serie en un valor inferior? Supongamos que tenemos un número de serie 2000042601. El número mayor que le podemos añadir para permitir que sea mayor que él es $2^{31}-1$ (2147483647). Para hacer la suma más sencilla, usemos en su lugar 2000000000. Así obtenemos 4000042601. El siguiente paso es dar los pasos en anillo más allá del cero. Para Mantener el concepto de numeración usando la fecha, tomamos 200004261 como el siguiente número de serie. Ahora tenemos el formato de nueve dígitos. Sencillo, ¿verdad?

Por supuesto, ha de esperar a que los secundarios obtengan (y usen) las zonas con el número de serie cambiado. Debe comprobarlos para llevar a cabo el cambio con éxito.

Finalmente obtenemos los siguientes pasos:

1. Incremente su número de serie en 2000000000 (diez dígitos)
2. Espere a que los secundarios hagan la transferencia de zona
3. Compruebe (todos) los secundarios con una de las siguientes órdenes


```
dnsq soa <dominion> <ip del secundario>
dig @<ip del secundario> <dominio> soa
nslookup -type=soa <dominio> <ip del secundario>
```
4. Establezca el número de serie con el valor de nueve dígitos
5. repita los pasos 2. y 3.

Quizá quiera obtener *información adicional sobre los números de serie*.

8 Comprobación de configuraciones DNS y diagnóstico de problemas

(nota: comentario de por qué *dnsq* y *dnsqr* son preferibles a *nslookup*)

9 Rendimiento

9.1 uso de memoria de dnscache

dnscache reserva la memoria caché que necesita en el momento de su arranque. No verá al proceso *dnscache* aumentar de tamaño. Puede establecer el parámetro `CACHESIZE` a cualquier valor que desee (la FAQ oficial explica cómo medir su efectividad). Para cualquier configuración que vaya más allá del uso "normal" de una única estación de trabajo, puede ser una buena idea pensarse lo de incrementar `CACHESIZE`.

La variable de entorno `DATALIMIT` que se usa en el guión de ejecución estándar es un mecanismo de seguridad. Se usa para evitar que el proceso comience a crecer sin control en el improbable caso de que haya bugs técnicos o lógicos que hagan crecer a *dnscache*. Las PUF (preguntas de uso frecuente, FAQ) proporcionan un ejemplo de cómo aumentar el tamaño de la caché a 100 MB. Si desea configurar otro valor (puede bastar con 2 MB para una empresa pequeña con correo y acceso web a través de una línea de 64 k), tendrá que decidir cómo configurar exactamente `DATALIMIT`.

Aunque nadie posee la fórmula para calcular el valor exactamente, he aquí 3 trucos que proceden de la lista de correo:

«El valor más pequeño que funcione. *dnscaché* reservará espacio para su caché al comienzo. Si `DATALIMIT` es demasiado pequeño, la reserva fallará. Después de la reserva inicial, ya no deseará que *dnscache* siga creciendo; `DATALIMIT` evita que ello suceda. Tal crecimiento sucedería sólo como resultado de un error de programación, y por tanto es improbable; `DATALIMIT` es un extra, una medida de protección por si acaso». (Paul Jarc en *este mensaje*)

Dan ha dicho: «Si desea experimentar con una caché de mayor tamaño, asegúrese de mantener el valor de `-d` un par de megabytes por encima del tamaño de la caché».

Uwe Ohse: «El valor `-d` es algo mayor que `CACHESIZE` simplemente porque *dnscache* necesita "cierta" cantidad de memoria además de la caché».

Para ajustar el tamaño de la caché tiene que monitorizar las líneas de "stats" que *dnscache* inserta en su fichero de registro o bitácora. Para ello Markus Stumpf ha creado un fichero separado usando el siguiente truco de multilog:

```
exec setuidgid dnsllog multilog t s250000 n20 ./main '-*' +'* stats * * *' =./dnsstatus
```

10 Instalación de paquetes

introducción a la instalación de fuentes frente a paquetes, `conf-home`

10.1 Instalación manual de fuentes

10.1.1 daemontools

Obtenga *daemontools* de <http://cr.yip.to/daemontools/install.html>. La instalación es francamente sencilla:

```
gunzip daemontools-0.70.tar.gz tar -xf daemontools-0.70.tar
cd daemontools-0.70
make
mkdir /service
```

Identifíquese como root e instale el paquete:

```
make setup check
```

Sólo resta asegurarse de que svscan se ejecuta en el proceso de arranque del sistema. Baste una indicación: svscan y supervise se instalan en /usr/local/bin, que no es parte del PATH en la mayoría de sistemas Unix.

Linux inicio en inittab frente a /etc/rc.d/ frente a inicios de tipo sysv

***BSD** En los *BSD, añade estas líneas a su /etc/rc.local:

```
echo -n "daemontools " PATH=$PATH:/usr/local/bin
svscan /service &
```

Solaris

10.1.2 ucspi-tcp

Si desea ejecutar axfrdns o axfr-get, necesita ucspi-tcp. Obténgalo de <http://cr.yp.to/ucspi-tcp/install.html>. Nuevamente la instalación es en verdad sencilla:

```
gunzip ucspi-tcp-0.88.tar.gz tar -xf ucspi-tcp-0.88.tar
cd ucspi-0.88
make
```

Identifíquese como root una vez más para la instalación:

```
make setup check
```

Es todo. No se precisa configuración.

10.1.3 Instalar djbdns

Obtenga djbdns de <http://cr.yp.to/djbdns/install.html>. El proceso de instalación es una vez más muy sencillo:

```
gunzip djbdns-1.02.tar.gz tar -xf djbdns-1.02.tar
cd djbdns-1.02
make
```

Y, oh maravilla, identifíquese como root para la instalación:

```
make setup check
```

10.2 Uso de RPMs

Lo siguiente es obra de Bennett Todd; Trabajamos con Red Hat Linux la mayor parte del tiempo, y a veces con Solaris: así pues, ilustraremos el procedimiento de instalación usando las herramientas que prefiero, entre ellas los guiones de inicio y los paquetes rpm que se encuentran en *rpm-packaging.tar.gz*. Los ficheros spec allí contenidos construyen rpms a partir de los fuentes (las URLs para bajarse los fuentes del sitio de djb se incluyen en los ficheros spec). También incluyen documentación, que elaboro haciendo una réplica de su website con *ftpcopy*, y luego empaquetándola con órdenes parecidas a

```
for pkg in djbdns daemontools ucspi-tcp;do tar cf - --exclude=*.gz $pkg* | bzip2 \
> /usr/src/redhat/SOURCES/${pkg}-docs.tar.bz2
done
```

Así pues, en la mayoría de las máquinas ejecuto dnscache como un resolutor local; su configuración es inmediata:

```
rpm -i daemontools-0.70-1.i386.rpm rpm -i djbdns-1.02-1.i386.rpm
useradd -d /no/home -s /no/shell dnscache
useradd -d /no/home -s /no/shell dnslog
dnscache-conf dnscache dnslog /etc/dnscache
ln -s /etc/dnscache /service
$EDITOR /etc/resolv.conf # set nameserver to 127.0.0.1
```

10.3 Uso de ports en OpenBSD 2.9 y anteriores

On OpenBSD 2.9 y anteriores, la installation era más sencilla usando la colección de ports. Si tiene el *ports.tar.gz* del CD1 desempaquetado en /usr, no olvide que OpenBSD no trae de serie referencia a djbdns, tiene que actualizar el CVS de su árbol de ports antes de instalar. Asegúrese de que ha configurado correctamente CVSROOT y CVS_RSH (lea <http://www.openbsd.org/anoncv.html> si no tiene claro lo que ello significa).

```
cd /usr cvs up -PA ports
cd ports/net/djbdns
make
su
make install
```

Estas órdenes obtendrán, compilarán e instalarán djbdns, daemontools y ucspi-tcp por Usted. Hay un paquete creado en /usr/port/packages/, que puede usar para instalar djbdns en otras máquinas usando `pkg_add`.

10.4 Uso de Ports en FreeBSD

El procedimiento en FreeBSD es básicamente el mismo que en OpenBSD, suponiendo que previamente haya instalado el árbol de ports.

```
su cd /usr/ports/sysutils/ucspi-tcp
make install
```

```

cd /usr/ports/sysutils/daemontools
make install
cd /usr/ports/net/djbdns
make install

```

Esta operación obtendrá, compilará e instalará djbdns, daemontools y ucspi-tcp por Usted. Habrá un paquete creado en /usr/ports/packages/ que puede usar para instalar djbdns en otras máquinas usando pkg_add.

10.5 Uso de ? en Solaris

10.6 Guión de inicio estilo SysV

En sistemas tipo SysV, preferirá tener un guión de inicio de estilo SysV seguramente. He aquí uno:

```

#!/bin/sh #
# chkconfig 2345 55 45
# svscan arranca y detiene el demonio svscan
#
# descripcion: svscan es un gestor de servicios de proposito
# general, que se encarga de la ejecución de
# demonios y asegura que estén operativos.
# nombre proceso: svscan

# Biblioteca fuente de funciones
# . /etc/rc.d/init.d/functions #rh6
# . /etc/init.d/functions #rh7

SERVICESDIR="/usr/local/service"
BINDIR="/usr/local/bin"
LOCKDIR="/var/lock/subsys"
PATH=$PATH:$BINDIR

-f$BINDIR/svscan]|

exit 0
-d$SERVICESDIR]|

exit 0

RETVAL=0

# Veamos cómo nos llamamos
case "$1" in
start)
# Start svscan.
action 'Arrancando svscan:' "/bin/sh -c '( cd $SERVICESDIR && exec $BINDIR/svscan ) &'"
RETVAL=$?
$RETVAL -eq 0]&&touch$LOCKDIR/svscan
;;
stop)
#Stopsvscan.
echo -n"Deteniendosvscan:"

```

```

killprocsvscan
RETVAL = $?
echo
$RETVAL -eq 0 ] && rm -f $LOCKDIR/svscan

# Detener los servicios supervisados. Lo hacemos en ultimo lugar,
# para que svscan no los reinicie.
x="$allownullglob expansion"
allownullglob expansion=1
for service in $SERVICESDIR/*
do
$BINDIR/svok $service || continue
action "Finaliza supervision del servicio 'basename $service':" "$BINDIR/svc -x
$service"
-k$service]&&action"Finalizasupervisiondelserviciodelog'basename$service' : ""$BINDIR/svc -
x$service/log"
action"Stoppingservice'basename$service' : ""$BINDIR/svc - d$service"
-k $service ] && action "Se detiene el servicio de log 'basename $service':"
"$BINDIR/svc -d $service/log"
done
allownullglob expansion="$x"
;;
status)
status svscan
RETVAL=$?
x="$allownullglob expansion"
allownullglob expansion=1
for service in $SERVICESDIR/*
do
$BINDIR/svstat $service
done
allownullglob expansion="$x"
;;
restart|reload)
$0 stop
$0 start
RETVAL=$?
;;
)
echo "Uso: svscan {start|stop|restart|reload|status}"
exit 1
esac

exit $RETVAL

```

11 uso de los guiones add-* para la creación del fichero de datos de tinydns

```
cd /etc/tinydns/root ./add-host maquina1.example.com 1.2.3.1
./add-host maquina2.example.com 1.2.3.2
./add-host maquina3.example.com 1.2.3.3
```

Si example.com está delegado a su servidor tinydns adecuado, la orden `dnsip maquina1.example.com` debería devolverle ya como resultado 1.2.3.1. Observe que ejemplo.com debe delegarse a a.ns.example.com con su IP, para que todo esto funcione. Si quiere que maquina1 sea accesible también como `www.example.com`, simplemente haga

```
cd /etc/tinydns/root ./add-alias www.example.com 1.2.3.1
```

Definamos igualmente maquina2 como nuestro servidor de correo para example.com:

```
cd /etc/tinydns/root ./add-mx example.com 1.2.3.1
```

Tinydns dará al servidor de correo el nombre `a.mx.example.com`. También puede definir más servidores de nombres:

```
cd /etc/tinydns/root ./add-ns example.com 1.2.3.3
./add-ns 3.2.1.in-addr.arpa 1.2.3.3
```

Tinydns llamará al segundo servidor `b.ns.example.com`.

12 enlaces útiles

Configurar djbdns para su uso con conexiones intermitentes por Matthias Andree
<matthias.andree@gmx.de>

13 Reconocimientos

Bennett Todd <bet@rahul.net> por comenzar este gran documento, la cantidad de esfuerzo invertido y su estupenda visión general de la estructura del documento. Gracias, Bennet. Ha sido en verdad divertido trabajar contigo.

A Dave Sill por su apoyo y sus ánimos, y por permitirme llamar a este documento "Mi vida con djbdns" en fanco homenaje a "Mi vida con qmail".

Joost van Baal <joostvb@xs4all.nl> por mejorar la descripción del registro NS.

Henning Brauer <hostmaster@bsws.de> por la sección sobre diseño para ISPs o proveedores de servicios de Internet (PSI) (exportar de un RDBMS a tinydns-data, y realización de notificaciones (Notify) para servidores secundarios, por las instrucciones de instalación y por el apéndice a.

Florent Guillaume <Florent.Guillaume@mail.com> por dos cambios en el guión dnsnotify para permitir que el notify fuese conforme al RFC 1996.

Jos Backus <josb@cncdsl.com> por el guión dnsnotify.pl, siempre actualizado.

Clemens <rabat@web.de> por las notas a la instalación en FreeBSD.

Richard Lucassen <lucasson@bigfoot.com> por mejorar el guión de inicio de estilo SysV.

David Pick <D.M.Pick@qmw.ac.uk> por añadidos a las notas de instalación de FreeBSD.

Frank Tegtmeier <fte@fte.to> por la sección acerca de los servidores raíz privados, la actualización desde BIND y varias otras contribuciones de las que ya he perdido la cuenta ;-))