UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Diseño y Construcción de Arquitectura de Alta Disponibilidad para el Monitoreo y la Administración de Dispositivos M2M.

INFORME DE ACTIVIDADES PROFESIONALES

Que para obtener el título de

Ingeniero en Telecomunicaciones

P R E S E N T A

Saúl Cortés Riquelme

ASESOR DE INFORME

Dr. Luis Andrés Buzo de la Peña

Ciudad Universitaria, Cd. Mx., 2016

AGRADECIMIENTOS.

Con infinita gratitud a mis padres, por su educación y ejemplo de esfuerzo, superación y sacrificio incondicional.

A mis hermanos, por su inspiración, apoyo y confianza.

Por último, mi entero reconocimiento a familiares, amigos, profesores, jefes y compañeros cuya motivación y experiencia han sido invaluables.

A todos ellos, GRACIAS.

CONTENIDO.

INTRODUCCIÓN.

El fácil acceso que el público tiene a la Internet y al uso de las tecnologías de la información ha provocado una evolución en la manera como la sociedad se comporta y también se comunica, pero muy particularmente en la forma como se desarrollan los negocios.

Con el paso del tiempo, la información se ha convertido en un bien invaluable para todos nosotros. En el lenguaje coloquial, la palabra información puede utilizarse como sinónimo de conocimiento, cultura, noticia, dato; en fin, se trata de una amalgama de significados sencillos y generalizados que dependerán del individuo que los cite para orientarlos hacia un campo determinado. En el terreno de las Telecomunicaciones siempre será relevante el aspecto técnico, refiriéndose a la información como un elemento cuantitativo que deriva en el flujo, mediante señales o signos, de un atributo intangible que circula a través de una red de comunicaciones.

Pero más allá del uso que suele darse a esta palabra, se puede profundizar mucho más en la importancia que tiene en otros entornos. La palabra información, en cierto sentido, desde siempre ha sido sinónimo de riqueza y poder. Contar con información que muchos consideren importante le da un valor proporcional al interés que tienen las personas por obtenerla. Este recurso entonces puede explotarse al grado de generar riqueza para su poseedor, con un manejo y explotación adecuados. Los responsables de administrar la información tienen la consigna de comprender la complejidad implícita en su generación, obtención, almacenamiento, protección, procesamiento, propagación, consulta y recuperación. Todo ello no es, en lo absoluto, gratuito y técnicamente sencillo, de tal manera que en adelante todo lo que se refiera a estos conceptos dentro de este documento girará en torno a la informática y las redes de datos.

La pérdida, corrupción o robo de la información por lo general supone varios problemas que se traducen en pérdidas económicas y de tiempo para cualquiera que tenga la desgracia de caer en ellos. Empresarialmente, la información se ha colocado en un lugar privilegiado como uno de los principales recursos, tan sólo por debajo del personal que la administra.

Es un hecho que en esta época somos prácticamente dependientes de todo tipo de sistemas computacionales cuya labor va, desde proporcionar los servicios para satisfacer las necesidades más básicas al gran grueso de la población, hasta administrar y controlar procesos muy complejos a nivel industrial y gubernamental. Pero, ¿qué sucedería si los elementos que realizan las tareas más críticas del sistema colapsan o resultan dañados por alguna falla de cualquier naturaleza? Las acciones para prevenir alguna eventualidad que suponga la caída de un sistema computacional son simples: diseñar, crear e implantar métodos que garanticen el buen rendimiento y la alta disponibilidad de los servicios en todo momento.

¿A qué se refiere la alta disponibilidad y qué implicaciones tiene? En palabras simples, se puede explicar como el acceso permanente de usuarios a un sistema para la consulta de información, manipulación de datos, ejecución de procesos y cualquier otro tipo de actividad permitida por dicho sistema. La alta disponibilidad asegura que todos los servicios que proporciona un sistema, en un escenario ideal, estén activos durante las 24 horas de los 365 días del año. Naturalmente ello implica aumentar y distribuir adecuadamente la infraestructura de cada eslabón, el desarrollo o configuración de todo el *software* necesario para hacerla funcionar y, además, robustecerlo con poderosas herramientas para la administración de sistemas operativos y de redes de datos.

Dentro de la terminología común que se maneja en torno a los sistemas computacionales pueden encontrarse varios conceptos interesantes. Cuando un programa se está ejecutando, se suele referir a él como un proceso. Un proceso que realiza su trabajo dentro de un sistema Linux es llamado demonio. Un demonio y los efectos que produce al efectuar sus actividades se define como un servicio. Un servicio se convierte en un recurso cuando se complementa con su medio ambiente operativo (archivos de configuración, datos que el usuario y el sistema proporcionan, mecanismos de red utilizados para lograr el acceso al mismo, entre otros). La caída de un sistema debida a la falla de un servicio y su posterior restablecimiento (*failover*), a menudo en un *hardware* de respaldo, sucede cuando el control del recurso en activo se mueve de una computadora a otra. Una configuración adecuada para lograr la conmutación de recursos y disponibilidad de los mismos carece de puntos únicos de falla.

Podemos llamar "servicios de misión crítica" simplemente a todos aquellos que demanden un diseño e implementación considerablemente robustos para permanecer activos todo el tiempo y que una ausencia de los mismos signifique pérdidas económicas, materiales o humanas para la empresa y sus clientes. Se podrían encontrar un sinnúmero de ejemplos de servicios de misión crítica, aunque pronto pueden imaginarse los más comunes que prestan los proveedores que dan acceso a la Internet, los bancos, telefonía fija y móvil, aeropuertos, bolsas de valores, operadores satelitales, etcétera. Imagine el caos que generaría la caída en las líneas de emergencia de un hospital o una estación de bomberos, una falla de comunicación con la torre de control de un aeropuerto o la pérdida de los servidores que manejan las transacciones en un banco, por mencionar sólo algunos.

Entrando en el contexto de este reporte, quizá a muchos nos ha pasado que al visitar un cajero automático para realizar una operación financiera, éste muestra la pantalla "Fuera de servicio". Naturalmente la molestia que esto produce motivará al usuario a buscar nuevas opciones que ofrezcan una mejor atención y una buena disponibilidad de todos sus recursos.

Los conceptos citados anteriormente son tan importantes que generan cierta calma y un alto grado de confianza ante cualquier fenómeno por improbable que resulte, como daños por *software* maligno, accidentes laborales, ataques físicos, desastres naturales, interrupción en las líneas de comunicación, etcétera. Saber que la información y las comunicaciones

están bien respaldadas deposita mucha más tranquilidad en las mentes de los administradores.

En otro sector, la seguridad informática es un área que tiene ocupados en todo momento a los especialistas de las empresas y todas las instituciones gubernamentales, de tal forma que las herramientas que permiten resguardar toda la información valiosa que sustenta su trabajo evolucionan todo el tiempo. Desafortunadamente dicho progreso se debe principalmente a los grandes avances que también hay para violar esa misma seguridad y, en otra instancia, al deterioro que el paso del tiempo, el descuido y los accidentes provocan en todas las cosas.

La evolución de los métodos más empleados para alcanzar altos porcentajes de disponibilidad va de la mano con el avance tecnológico, la infraestructura disponible y el presupuesto al que tienen acceso los interesados. En ese sentido, se busca siempre economizar al máximo, cuidando al mismo tiempo que no se le apueste toda la operación a un solo elemento, por muy confiable que éste sea.

Entre las técnicas mayormente utilizadas está la adquisición de equipos que cuentan con duplicidad en todos sus componentes esenciales. Servidores que poseen múltiples procesadores que se distribuyen el trabajo, sistemas de almacenamiento que utilizan varios discos duros para organizar y replicar datos –conocidos como RAID: *Redundant Array of Independent Disks*–, al menos dos tarjetas de red para comunicarse y fuentes de poder independientes; entre varias características más.

Por otro lado, existe una solución con bastante popularidad entre los sistemas de alta disponibilidad e implica la construcción de *clusters* para albergar una buena cantidad de computadoras de todos tipos y tamaños, reduciendo costos, aprovechando los recursos disponibles y favoreciendo al rendimiento óptimo.

La gran mayoría de seres humanos en la actualidad estamos íntima e involuntariamente ligados al término "*cluster*" y su existencia es incluso imprescindible para llevar a cabo muchas actividades que ahora forman parte de nuestra vida diaria. A lo largo de los capítulos que dan forma a este reporte se abordará con mayor holgura la definición de un *cluster* y las implicaciones que su diseño conlleva.

La madurez y la popularidad que han alcanzado los sistemas operativos Linux en el desarrollo de complejos sistemas informáticos nos hablan de la presencia que tienen alrededor del mundo las herramientas de código abierto (también conocidas como *open source*). Sus códigos están siendo continuamente mejorados y su innovación en sistemas operativos que utilizan *software* libre resulta útil para aficionados, estudiantes, académicos y desarrolladores de todo tipo. Empresas como Google, HP, IBM, Cisco y varias instituciones bancarias han apostado fuertemente a este proyecto y con extraordinarios resultados. Indudablemente el *software* libre tiende cada vez más a la estabilidad y compatibilidad con todos los desarrollos.

El presente informe resume el proyecto que fue realizado durante el primer semestre de 2014 por la compañía en la que ahora desempeño mis actividades laborales, en adelante La Empresa, en donde participé activamente en su planeación, diseño y puesta en marcha, así como en su administración, mantenimiento y mejoras posteriores a su conclusión.

El rubro de La Empresa, entre otras cosas, es la prestación de servicios de telemetría y monitoreo de dispositivos M2M que se comunican a través de la red celular por medio de un SIM (*Subscriber Identity Module*), tal y como lo hacen actualmente los *smartphones* para navegar por Internet, con la diferencia de que las líneas contratadas por La Empresa pertenecen a una red privada, segura y sin servicio de voz; únicamente están habilitadas para la transmisión de datos por la red celular de IP hasta el cliente.

¿Qué significa M2M?

Se han hecho varios intentos de proponer un solo significado a las letras M del acrónimo M2M: *Machine-to-Machine*, *Machine-to-Mobile*, *Machine-to-Man*, etcétera. No obstante, quizá el más usado por el grueso de personas cuyo trabajo está íntimamente ligado a este concepto es el primero: *Machine-to-Machine*.

Más allá del acuerdo que se tenga para interpretar las siglas M2M, es aún más importante entender lo que hay inmerso en un sistema con sus características. El alcance de M2M es, por naturaleza, elástico y los límites no siempre están claramente definidos.[1]

El rol de M2M es establecer las condiciones que permitan a un dispositivo intercambiar información bidireccionalmente con una aplicación por medio de una red de comunicaciones, de tal forma que el dispositivo o la aplicación puede actuar como la base para este intercambio –Figura I (a)–. En esta definición, la red de comunicaciones tiene un papel fundamental: la aplicación utilizada y el dispositivo, por sí solos, difícilmente pueden considerarse como un conjunto que tiene una relación M2M. Esta es la razón por la que M2M a menudo se utiliza como sinónimo abreviado para M2(CN2)M: *Machine-to-*(*Communication-Network-to-*)*Machine*.

En muchos casos, M2M involucra a un grupo de dispositivos similares interactuando con una única aplicación –Figura I (b)–. La gestión de flotillas es un ejemplo de ésta aplicación donde los dispositivos son, por ejemplo, camiones, y la red de comunicación es una red de telefonía móvil. En otras situaciones, como se muestra en la Figura I (c), los dispositivos en el grupo pueden no estar directamente interactuando con la aplicación por tener únicamente capacidades y permisos limitados. En este escenario, la relación está mediada por otro dispositivo (por ejemplo, una puerta de enlace o *gateway*) que permite de alguna forma establecer la comunicación. *Smart metering* es un ejemplo de una aplicación donde los dispositivos son medidores inteligentes y la comunicación se puede dar por una red móvil o la Internet pública.

[1] David Boswarthick, Omar Elloumi, Olivier Hersent. “M2M Communications: A Systems Approach”.

Sólo para tenerlo en cuenta, el término "M2M *area network*" ha sido introducido por el *European Telecommunication Standards Institute* (ETSI). Una red de área M2M ofrece conectividad a nivel de la capa física OSI entre los diferentes dispositivos M2M conectados a la misma red de área M2M, permitiendo así que los dispositivos M2M puedan tener acceso a una red pública a través de un *router* o *gateway*.

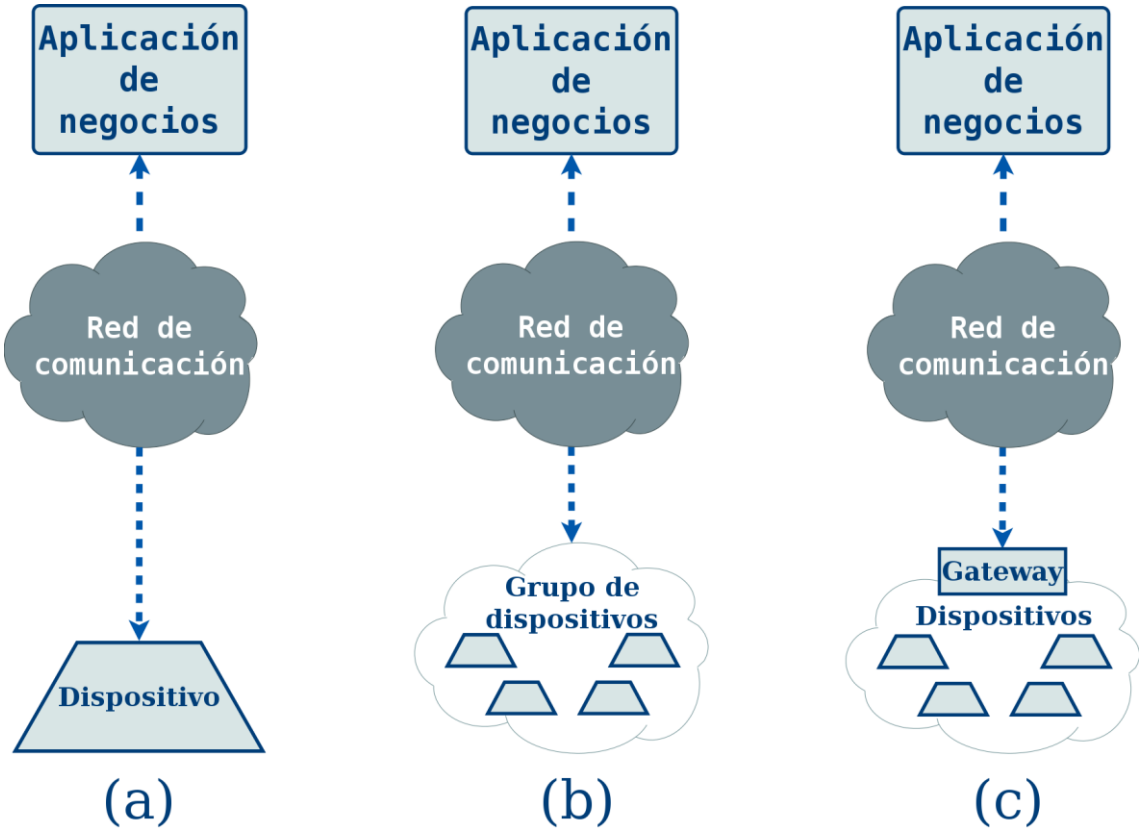

Figura I. Ejemplos de sistemas M2M.

El alcance del informe no contempla profundizar en el conocimiento de los sistemas M2M, sino ofrecer una solución al monitoreo y alta disponibilidad para el control de los mismos; por ello no se abundará mucho más en el tema. Cabe mencionar solamente que la evolución de los sistemas M2M se ha convertido, en años recientes, en un campo de negocios muy atractivo incluso para las empresas de mayor prestigio, quienes desarrollan soluciones específicas a las demandas que el mercado tiene para el control y automatización de equipos de comunicación o de entretenimiento, equipos de medición, aplicaciones en el campo de la medicina, equipos de control, microprocesadores instalados en muchos dispositivos del hogar, oficina o automóvil, complejos servidores en un *Data Center*, entre muchos ejemplos más, conviviendo siempre con información en tiempo real.

Así, el objetivo general del proyecto fue trabajar en un sistema redundante en comunicaciones WAN y, a nivel local, para los servidores que reciben y procesan la información proveniente de los dispositivos M2M conectados a la red celular, que posteriormente es almacenada y consultada en diferentes bases de datos. La solución ayuda enormemente al control, monitorización y registro estadístico principalmente de ATM (*Automated Teller Machine*), *vending machines* y TPV (Terminal Punto de Venta) o POS (*Point Of Sale*) que ofrecen servicios de carácter crítico y que a menudo involucran fuertes cantidades de dinero en inversión y logística, motivos por los cuales el cliente busca alternativas que maximicen las ganancias y la prosperidad de su negocio.

El reporte está dividido en cinco capítulos y a lo largo de ellos se describen los pasos que permitieron cumplir satisfactoriamente el objetivo, partiendo de un buen aprendizaje teórico, trabajo de laboratorio con simulaciones y equipos reales –pruebas que no se incluyen en este informe– y finalmente los resultados obtenidos al término del proceso.

En el documento sólo se contemplan los aspectos técnicos para la creación del sistema ya descrito, dejando de lado el aspecto económico, administrativo empresarial y las condiciones ambientales para instalar cualquiera de los elementos involucrados. No se incluye un modelo de negocios y tampoco un análisis de rentabilidad que sustente el uso de un sistema de alta disponibilidad. Tiene un enfoque técnico sobre un método para lograr que varios enlaces WAN se respalden automáticamente entre sí, alcanzando segmentos específicos de la red de un proveedor o de un cliente. También se enfoca en la mejora de un sistema tradicional de almacenamiento y consulta de información, compuesto por un *firewall* con una interfaz conectada a Internet, un servidor de base de datos donde se guarda la información para posteriormente ser explotada y un servidor web dentro de un área desprotegida para hacer consultas y operaciones a través de Internet. Tampoco se mencionan las reglas de seguridad necesarias para el *firewall*.

Se omitirán, también, temas de diseño para elegir los componentes del *hardware* que integra un *cluster*. La selección de los elementos mencionados en el Capítulo 4 y sus características estuvo a cargo de un área técnica distinta.

De acuerdo con las limitaciones señaladas, el esquema base a seguir es el mostrado en la Figura II.

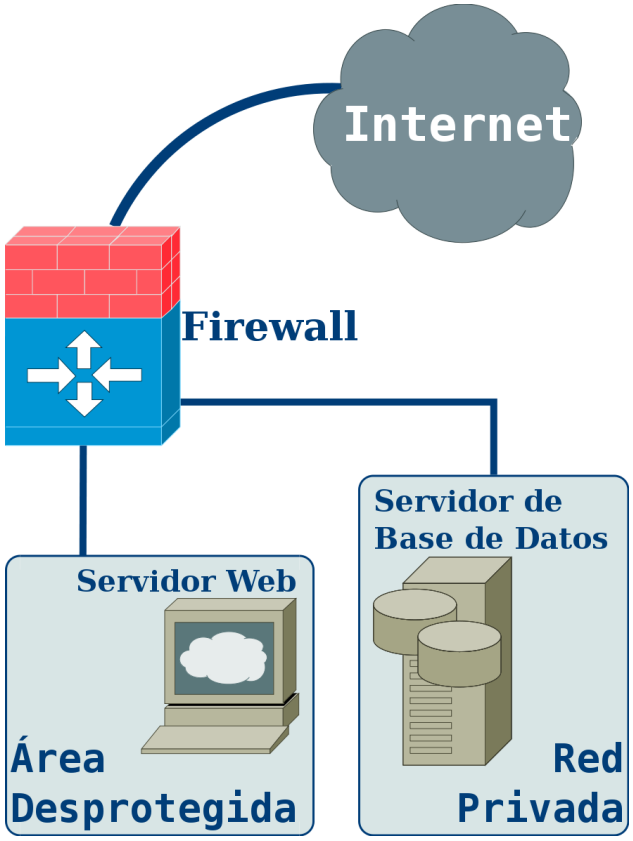

Figura II. Arquitectura tradicional de almacenamiento y consulta de información por Internet.

Capítulo 1.

Clusters en Linux.

1.1 Definición de *cluster*.

Lo primero que debe establecerse claramente para comenzar a dar forma a este trabajo es una definición que sea simple, pero a la vez completa, de un *cluster* en redes de datos. Buscando en la Internet y la bibliografía del tema es posible encontrarse con muchas definiciones, algunas más técnicas y formales que otras, dependiendo del enfoque del autor. A continuación se enuncian algunas de ellas:

"Un *cluster* es una colección de sistemas informáticos estrechamente relacionados, proporcionando un servicio en común o ejecutando una aplicación paralela común".[2]

Es un "sistema que puede ser usado como un único recurso de computación que emplea un sistema local de cómputo y que comprende un conjunto de computadoras independientes y una red que las interconecta".[3]

Un *cluster* es "una colección de computadoras (homogénea[4] o heterogénea[5]) que están conectadas dentro de una red privada con altas tasas de transferencia de datos que les permite compartir y utilizar sus recursos como una única computadora".[6]

Se define al *cluster* como un "conjunto de computadoras independientemente operacionales, integradas por medio de una red interconectada y soportada por *software* accesible para el usuario, quien organiza y controla las tareas computacionales en tiempo real que pueden cooperar en un programa de aplicación común o de carga de trabajo".[7]

"Un *cluster* es una colección de elementos computacionales que cooperan libremente acoplados, a los que suele referirse como nodos. Las fallas en un *cluster* no son observadas instantáneamente o simultáneamente por todos los nodos. En cambio, las fallas ocurren asíncronamente y son observadas estocásticamente e independientemente por los nodos de un *cluster*".[8]

A partir de la información anterior, se puede elaborar una definición sencilla que reúna los aspectos más significativos mencionados por cada autor. Siendo así, la definición de

[2] Robert W. Lucke. "Building Clustered Linux Systems."
[3] Karl Kopper. "The Linux Enterprise Cluster: Build a Highly Available Cluster with Commodity hardware and Free Software."
[4] Se entiende por una red homogénea a aquella en la cual todos sus nodos poseen el mismo *hardware* y funcionan bajo el mismo sistema operativo.
[5] En una red heterogénea los nodos pueden tener *hardware* distinto y trabajar con diferentes sistemas operativos.
[6] Enrique Vargas, Joseph Bianco, David Deeths: "Sun Cluster Environment: Sun Cluster 2.2".
[7] Charles Bookman. "Linux Clustering: Building and Maintaining Linux Clusters".
[8] http://www.linux-ha.org/

cluster que se utilizará formalmente para este documento es:

"Un *cluster* involucra a un sistema de computadoras que están comunicadas a través de una red, trabajando en conjunto y ofreciendo a los usuarios un alto rendimiento y una alta disponibilidad de los diferentes servicios que administra, funcionando como un único recurso de procesamiento computacional".

La Figura 1.1 muestra el esquema de un *cluster* genérico de dos nodos. Es muy recomendable que la comunicación entre los nodos que componen el *cluster* se lleve a cabo de manera independiente a la infraestructura que soporta la red corporativa, es decir, con interfaces de red extra y conectadas entre sí directamente o a través de un *switch* (o VLAN) dedicado exclusivamente para los nodos del *cluster*.

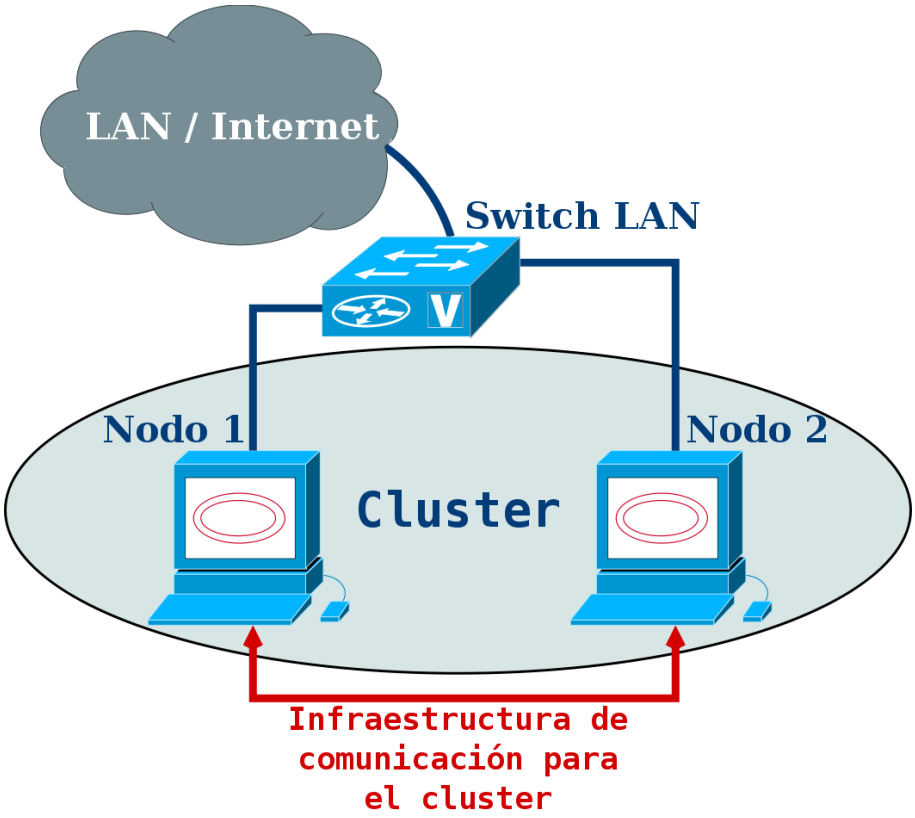

Figura 1.1. Esquema general para un *cluster* de dos nodos.

Además de la definición anterior, algunos de los fundamentos básicos deseables en un *cluster* son los siguientes:

- Los usuarios no deben saber que están utilizando un *cluster*. Para ellos todo su procedimiento de consulta y de trabajo debe ser transparente.
- Los nodos que funcionan dentro de un *cluster* no deben saber que forman parte del mismo. En otras palabras, el Sistema Operativo no necesita ser modificado

para adaptar un nodo a un *cluster*, mientras que la falla de un elemento no debe tener efecto en los otros nodos que trabajan dentro del mismo *cluster*. Cualquier computadora puede ser reiniciada o retirada completamente del *cluster* sin que las demás resulten afectadas.

- Complementando el punto anterior, cualquier computadora dentro de un *cluster* o cualquier computadora que dependa del *cluster* para operar, en condiciones normales, puede ser reiniciada sin la necesidad de reiniciar todo el *cluster*.
- Las aplicaciones que se ejecutan en un *cluster* no tienen por qué saber que están trabajando dentro de uno. Si una aplicación –especialmente de misión crítica– debiera ser modificada para funcionar dentro de un *cluster*, entonces dicha aplicación ya no estaría utilizando al *cluster* como único recurso de cómputo unificado.
- Los demás servidores en la red no deben saber que están dando servicio a un nodo que pertenece a un *cluster*. Los nodos que pertenecen a un *cluster* deben ser capaces de hacer solicitudes a los servidores de la red tal como lo haría cualquier otro cliente común y corriente. Los servidores de la red no deben ser reconfigurados o "parchados" de forma alguna para soportar solicitudes de los nodos de un *cluster*.

Por supuesto, cabe la posibilidad de que alguno de miembros del *cluster* se encuentre fuera de línea o incluso pueda quedar inesperadamente inhabilitado para continuar con su trabajo. Si esto sucediera, el módulo balanceador de carga o el nodo encargado de la administración del *cluster* debe ser capaz de remover al nodo caído y alertar a la(s) persona(s) responsable(s) de la administración del sistema. Una vez más, se hace énfasis en que los miembros de un *cluster* deben ser independientes uno del otro, así que en ningún momento deben verse sobrecargados por la falla de uno sus nodos compañeros.

Los *clusters* pueden ser diseñados de múltiples formas –algunas arquitecturas de *clusters* son prácticamente basadas en *hardware*, mientras que otras son una combinación de *hardware* y *software*– y la tecnología que permite su implantación ha ido evolucionado en función de la necesidades del mundo actual; desarrollos tales como aplicaciones de cómputo avanzadas, *software* de servicios críticos, servidores web y banca electrónica, hasta llegar a la convivencia con bases de datos de alto rendimiento y sistemas de archivos diversos. Hasta el día de hoy, los *clusters* tienen un papel vital en la solución de problemas en dichas áreas.

La necesidad y la idea de utilizar *clusters* surgen como resultado de la aparición de algunas tecnologías que permiten la convivencia de microprocesadores económicos de alto rendimiento y redes de datos de alta velocidad, así como el desarrollo de herramientas en *software* capaces de soportar cómputo distribuido o paralelo.

1.2 Tipos de *clusters*.

Aunque se ha llegado a una definición muy genérica sobre las funciones de un *cluster*, aún no es posible precisar algo más referente a su arquitectura o clasificación; mucho menos sobre su funcionamiento base y qué elementos y procesos están involucrados en el mismo.

Durante los siguientes puntos se explicará de forma breve cuáles son los tipos de *clusters* comúnmente utilizados hasta ahora y en qué clasificación se posicionan en base a sus características.

1.2.1 Clasificación por aplicación.

De acuerdo a su uso, su configuración y la clase de servicios que ofrecen, se puede hacer una clasificación como la siguiente:

Cluster de alto rendimiento.

Su principal característica es ejecutar múltiples tareas que dispongan de una gran capacidad de memoria y procesamiento. Sus recursos están siendo consumidos permanentemente por estos procesos y sus aplicaciones son principalmente científicas.

Cluster de alta disponibilidad.

Su principal objetivo es alcanzar la máxima disponibilidad de los servicios para los cuales fue diseñado. El diseño en *hardware* busca evitar a toda costa la existencia de un único punto de falla (SPOF: *Single Point Of Failure*) que comprometa la actividad del sistema. El *software* con el que operan es lo suficientemente bueno para detectar caídas en algún servidor, en parte del mismo o dentro de la propia red y ofrece soluciones de recuperación inmediatas. Su aplicación normalmente es comercial y suele utilizarse en empresas que prestan servicios de consulta y comunicación a muchos clientes.

Éste fue el tipo de *cluster* seleccionado para el proyecto.

Cluster de alta eficiencia.

Tienen la característica de ejecutar la mayor cantidad de procesos posible durante la menor cantidad de tiempo. Suelen emplear *software* para el procesamiento paralelo o distribuido (dependiendo de la aplicación) entre una gran cantidad de nodos. Sin embargo, la importancia de mantener tiempos de respuesta óptimos en la comunicación entre los nodos pasa a segundo término. Su aplicación suele estar en actividades científicas y de investigación, en donde se presta mucha más atención a los resultados finales que a la frecuente consulta de los datos.

Las particularidades de la clasificación anterior no implican una imposibilidad de diseñar un *cluster* que combine las características de cada uno. De hecho, dentro de las bondades que se tienen al implementar un *cluster* está precisamente esa capacidad para poder elegir y combinar diferentes propiedades de acuerdo a las necesidades existentes. Evidentemente, se deben considerar las adecuaciones en *hardware* y *software* que derivan de cada modificación.

1.2.2 Clasificación por funcionamiento.

Además del uso para el cual está destinado un *cluster*, habría que poner atención a la forma en la que desempeña sus tareas para clasificarlo de acuerdo a su funcionamiento. Básicamente se pueden considerar dos tipos:

Cluster activo-activo.

En este tipo de *cluster* todos los nodos están en funcionamiento permanente; los mismos recursos están activos en cada nodo y la carga es distribuida entre todos ellos. Para construir un *cluster* activo-activo debe considerarse que al menos dos equipos están proporcionando el mismo servicio y los clientes pueden conectarse a ellos de manera indistinta, sin notar diferencia alguna.

Las ventajas de esta configuración son, sin duda, el aumento proporcional en la capacidad de procesamiento de acuerdo al número de nodos. Sin embargo, habría que considerar el tipo de servicio que prestaría una configuración así, ya que no todas las aplicaciones están diseñadas para ser implementadas en un *cluster* activo-activo y, de lograrlo, suelen ser susceptibles de caer en configuraciones complicadas y más difíciles de mantener y analizar.

Cluster activo-pasivo.

Esta configuración únicamente permite a un nodo proporcionar el servicio completo a los usuarios. El resto de las máquinas permanecen pasivas y vigilando constantemente al nodo primario, en espera de una falla para tomar el control de sus tareas con base en la prioridad que tengan asignada.

Entre sus ventajas destaca la compatibilidad con prácticamente cualquier tipo de servicio, permitiendo una implantación relativamente sencilla, fácil de entender y de diagnosticar. Por otro lado, la capacidad de cómputo de datos se ve limitada por el máximo rendimiento del servidor en activo.

Grid Computing.

Podría definirse como un caso muy particular derivado de un *cluster* activo-activo. Se basa en la utilización de los recursos de diferentes computadoras que no necesariamente conviven dentro de la misma red, por lo cual pueden incluirse servidores geográficamente dispersos.

Esta configuración está diseñada para utilizar los recursos de varias computadoras en línea (incluso cientos o miles de ellas) que en ese momento se encuentran en reposo o que simplemente sus capacidades no están siendo aprovechadas por completo. Todo ese potencial de procesamiento se puede utilizar para incrementar la capacidad de cálculo de un conjunto de máquinas dedicadas completamente a la realización de una tarea en particular.

El modo de operación más común en este tipo de *clusters* es captando el mínimo de recursos en función del porcentaje de ocupación, por parte del usuario, de su capacidad de procesamiento; o cuando el nodo está en reposo o completamente en desuso, pero en línea.

1.3 Multi-Site *clusters* (Geo *clusters*).

Resulta cómodo pensar que un *cluster* localmente establecido puede llegar a resolver prácticamente todos los inconvenientes que genera la falta de fiabilidad tanto en el *hardware* como en el *software* de los dispositivos. Sin embargo, es de lo más común encontrar grandes compañías que tienen presencia en más de una ciudad e incluso en más de un país o continente. Probablemente la primera duda que viene a nuestras mentes es la forma en cómo todos estos puntos se comunican estando geográficamente distantes unos de otros. Es necesario, entonces, hacer referencia a ciertas herramientas y medios que nos permitan armar una solución completa para estos casos, cuyo panorama irá tomando forma más adelante.

Dejando de lado el punto de cómo establecer una comunicación segura y confiable entre dos sitios distantes, enfoquemos nuestra atención al hecho de coordinar diferentes procesos que manejan los servidores en esos lugares.

Tener múltiples instalaciones dostribuidas geográficamente con un *cluster* local en cada una y que todas ellas tengan la coordinación suficiente para seguir comportándose como un solo ente, sí es posible. La conmutación por falla entre estos *clusters* distantes es coordinada por una entidad de mayor nivel llamada *booth*. Para tener una idea más clara sobre el concepto, conviene hacer un mapeo de dicho escenario en donde cada sitio se considera como un nodo en un *cluster* tradicional y cada uno de estos elementos es administrado por el mecanismo *booth* ya mencionado.

Repasemos algunos conceptos esenciales en una arquitectura de *cluster multi-site*:

- *Ticket*. Los *tickets* son, esencialmente, atributos de un *cluster* extendido. Garantiza el derecho a tener el control de ciertos recursos en un sitio específico.

Los recursos están ligados a un determinado *ticket* por medio de dependencias. Sólo si el *ticket* está disponible en el sitio, los recursos asociados a él son iniciados. Viceversa, si el *ticket* es revocado, los recursos que controla serán detenidos. Un *ticket* sólo puede ser poseído por un sitio a la vez e inicialmente ninguno de ellos lo tiene en su poder, hasta que llegar a un consenso.

- CTR (*Cluster Ticket Registry*). Es una entidad de alto nivel que coordina *clusters* geográficamente dispersos. Este mecanismo sirve para administrar *clusters* superpuestos garantizando que los recursos del *cluster* estarán disponibles a través de los diferentes sitios. Esto se logra con el uso de *tickets*, que son tratados como un dominio de *failover* entre los sitios, en caso de que uno de ellos colapse.
- *Booth*. Es la instancia que administra la distribución de los *tickets* y, por ende, la conmutación de los procesos entre los sitios de un *cluster* multi-*site*. Cada uno de los árbitros y *clusters* participantes deben ejecutar un servicio llamado *boothd* (*booth daemon*), éste conecta con los demás servicios ejecutándose en otros sitios e intercambia detalles de conectividad. Una vez que el *ticket* es asignado a un sitio, el mecanismo *booth* lo administrará automáticamente: si el sitio que mantiene el *ticket* queda fuera de servicio, evidentemente pierde el derecho a conservarlo y los sitios activos restantes votarán para decidir cuál de ellos lo obtendrá. Como protección extra, el sitio que perdió comunicación con el resto debe renunciar al *ticket* después de un cierto tiempo de espera.
- Árbitro. Como se ha mencionado anteriormente, los sitios se comunican entre todos por medio del proceso *boothd*. Si se tiene una organización con un número de sitios par, se necesitará una instancia adicional para alcanzar un consenso sobre las decisiones tales como el *failover* de los recursos entre ellos. En este caso, se agrega al menos un árbitro que se ejecute en una locación diferente. Los árbitros son simples máquinas que ejecutan una instancia *booth* en un modo especial. Como todas las instancias *booth* se comunican entre sí, los árbitros ayudan a tomar decisiones más confiables acerca de otorgar o revocar *tickets*.

  Un árbitro es especialmente importante en un escenario con sólo dos sitios. Por ejemplo, si un sitio A queda imposibilitado para comunicarse con un sitio B, existen dos posibles causas para eso:

  - Una falla de red entre A y B.
  - El sitio B está fuera de servicio.

  Sin embargo, si el sitio C (el árbitro) aún puede comunicarse con el sitio B, entonces se infiere que dicho sitio debe estar aún en servicio y ejecutándose. De esta manera se pueden tomar decisiones más acertadas. Ver Figura 1.2.

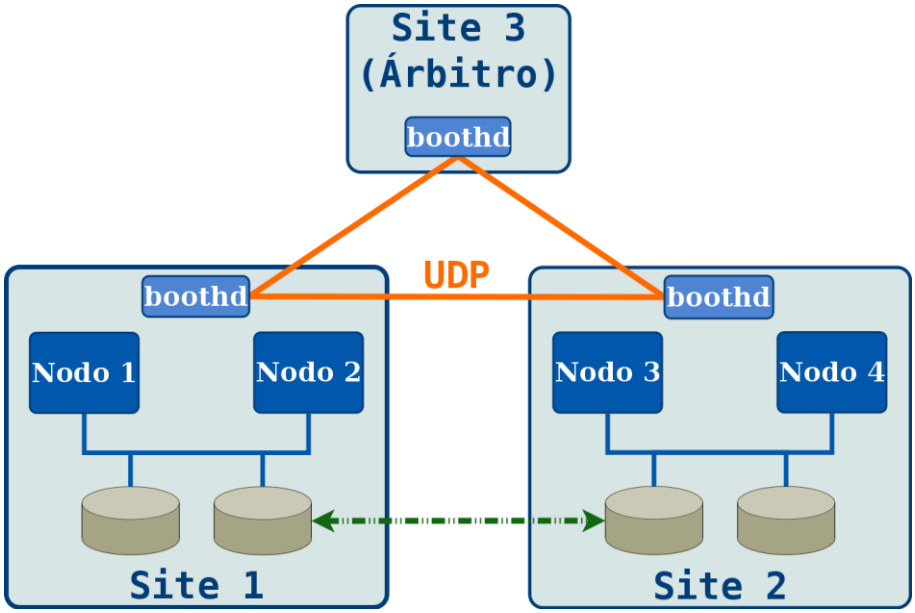

Figura 1.2. Arquitectura de un geo *cluster* con 3 *sites*.

- *Dead Man Dependency*. Un sitio solamente puede activar sus recursos de forma segura si él mismo puede comprobar que en los demás lugares están inactivos. Sin embargo, después de que el *ticket* es revocado, puede pasar un largo tiempo mientras todos los recursos asociados a ese *ticket* son detenidos limpiamente, especialmente en el caso de recursos funcionando en cascada. Para agilizar ese proceso, el administrador del *cluster* puede configurar una política de pérdida (*loss-policy*) junto con las dependencias del *ticket* en caso de que éste sea revocado del sitio. Es ahí donde interviene la característica de *fencing*, famosa dentro de las terminologías empleadas en los *clusters*, la cual permite aislar los recursos relacionados con el *ticket* e incluso apagar completamente el nodo en cuestión para evitar mayores problemas.

Capítulo 2.

Conceptos básicos de redes.

En este capítulo se abordará la teoría suficiente para entender y avanzar en la explicación del proyecto. Son temas importantes que ayudan ampliamente a manejar todas las herramientas que se utilizaron y que tienen fundamento en conceptos de redes de datos que tal vez la mayoría de estudiantes y egresados dedicados a dicha área deberíamos conocer. La secuencia en la que se exponen los temas está pensada para llevar el orden de aprendizaje más parecido al ideal.

2.1 Redes IP.

Tradicionalmente se define a una red como un conjunto de dispositivos que se interconectan o se comunican entre sí con el fin de compartir información. Sin embargo, con el increíble despunte tecnológico que se ha logrado tener en la última década, es probable que esa definición tan simple se quede corta para abarcar el total de características y posibilidades con las que se cuentan ahora, aunque finalmente la esencia podría seguir siendo la misma. Desarrollos que nos llevan cada vez más a vivir en un mundo donde todos los aparatos y accesorios que utilizamos estén permanentemente conectados a una nube virtual de información, con algún fin aparentemente útil, han dado origen a un concepto que se había planteado desde hace varios años: IoT (*Internet of Things*) o Internet de las cosas.

A pesar de la diversificación de las aplicaciones que pudieran modificar la manera en cómo se concibe una red de datos, los conceptos tradicionales son suficientes para cubrir el marco teórico que necesitamos.

2.1.1 PAN (*Personal Area Network*).

Las Redes de Área Personal permiten a los dispositivos comunicarse en un rango dentro del alcance geográfico de una persona. Un ejemplo muy recurrente es la red inalámbrica que conecta a una computadora con sus dispositivos periféricos (teclado, ratón, impresora, PDA, etcétera) a través de algunas tecnologías como Bluetooth o ZigBee.

Éste tipo de red no es relevante para nuestro propósito.

2.1.2 LAN (*Local Area Network*).

Una Red de Área Local es una red privada que opera dentro y cerca de una misma construcción, como puede ser una casa, una oficina o una fábrica. Son ampliamente socorridas para conectar computadoras personales, dispositivos periféricos y aparatos electrónicos de clase "*smart*" para que puedan compartir recursos e intercambiar información. Cuando una LAN es utilizada por una compañía, recibe el nombre de red empresarial.

Lo más común es encontrar redes de esta clase con una infraestructura tipo Ethernet, compuesta por *routers*, *switches* y *bridges*, combinada con puntos de acceso inalámbricos que cumplen con el estándar IEEE 802.11 (WiFi).

2.1.3 MAN (*Metropolitan Area Network*).

Las Redes de Área Metropolitana típicamente cubren una ciudad. Los ejemplos más conocidos de MAN son las redes de televisión por cable disponibles en algunas ciudades. A menudo se interconectan con enlaces dedicados a través de redes MPLS, *Frame Relay* u otro protocolo de encapsulamiento disponible.

Desarrollos recientes en redes inalámbricas de acceso a Internet han resultado en otra MAN, que la IEEE estandarizó como 802.16 y es popularmente conocida como WiMAX.

2.1.4 WAN (*Wide Area Network*).

Las Redes de Área Amplia comprenden un área geográfica muy grande, incluso un país o continente entero.

En la mayoría de las WAN, las subredes se componen de dos elementos principales: las líneas de transmisión y los elementos de conmutación. Las líneas de transmisión transportan el flujo de bits entre las máquinas. Pueden estar hechas de cables de cobre, fibra óptica o incluso enlaces de radio. La mayoría de compañías no tienen líneas de trasmisión propias; en lugar de eso rentan infraestructura a compañías de telecomunicaciones.

Por otro lado, los elementos de conmutación son computadoras especializadas que conectan dos o más líneas de transmisión. Cuando la información arriba por una línea entrante, el elemento de conmutación debe seleccionar la línea de salida por la que reenviará la información. Estos elementos han recibido varios nombres a lo largo de los años; actualmente el nombre "*router*" es el más utilizado.

2.1.5 Modelos de referencia.

Una vez que se ha resumido la clasificación de redes más general que podemos encontrar, toca el turno de revisar la arquitectura mediante la cual se puede analizar el procesamiento de la información en un dispositivo de la red y cómo viaja a través de la misma. Se discutirán las dos más importantes: el modelo de referencia OSI y el modelo de referencia TCP/IP. Aunque los protocolos asociados al modelo OSI ya no son utilizados con frecuencia, el modelo por sí mismo es bastante bueno y muy válido. Por otro lado el modelo TCP/IP tiene las propiedades contrarias: el modelo por sí mismo no es muy referenciado pero sus protocolos son ampliamente utilizados. Revisaremos brevemente la estructura de ambos.

2.1.5.1 OSI (*Open Systems Interconnection*).

Este modelo está basado en una propuesta desarrollada por la ISO (*International Standards Organization*) como un primer paso hacia la estandarización internacional de los protocolos utilizados en varias capas. También es llamado ISO OSI porque concuerda con los sistemas de conexión abiertos.

El modelo OSI tiene siete capas. Los principios que fueron aplicados para llegar a estas siete capas pueden ser brevemente resumidos como sigue:

1) Una capa debe ser creada donde un nivel diferente de abstracción es requerido.
2) Cada capa debe desempeñar una función bien definida.
3) La función de cada capa debe ser elegida con miras hacia la definición de protocolos estandarizados internacionalmente.

Una representación de la pila OSI y el procesamiento de la información compartida entre dos nodos que se comunican a través de un par de *routers* se muestran en la Figura 2.1.

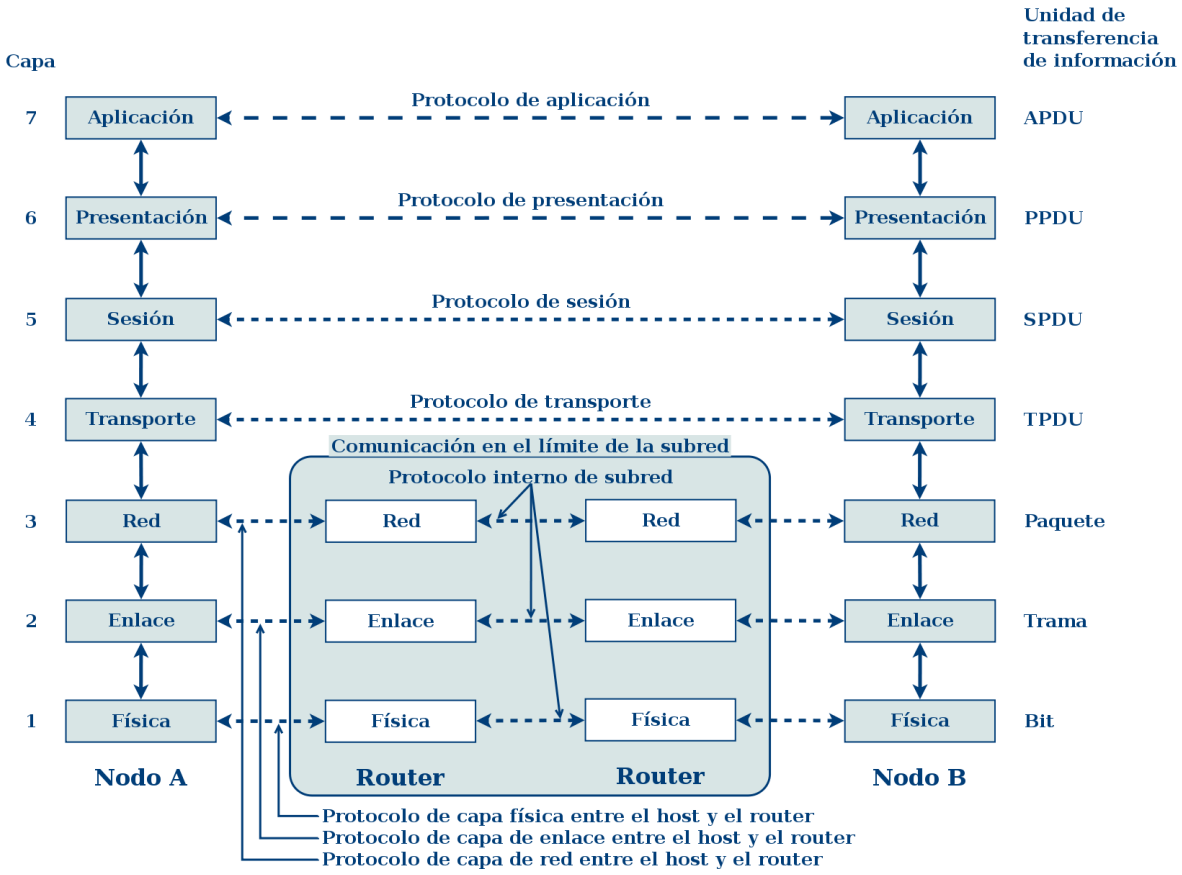

Figura 2.1. Comunicación entre dos dispositivos usando el modelo de referencia OSI.

Capa física.

La capa física se ocupa de la transmisión de bits "crudos" sobre un canal de comunicaciones. Los problemas de diseño tienen que ver con la seguridad de que en un lado se envíe un bit en 1 y, luego de ser transportado por los medios físicos de transmisión, se reciba un 1 y no un 0 en el destino. Las preguntas típicas aquí son: ¿qué tipo de señales eléctricas deben ser utilizadas para representar un 1 y un 0? ¿Cuántos nanosegundos debe durar un bit? ¿La transmisión puede suceder simultáneamente en ambas direcciones? ¿Cómo es establecida la conexión inicial?

Capa de enlace de datos.

La tarea principal de la capa de enlace de datos es transformar fácilmente una transmisión "cruda" de bits en una línea que esté libre de errores de transmisión no detectados. Lo hace mediante el enmascaramiento de errores reales de tal manera que la capa superior no pueda verlos. Lleva a cabo su tarea haciendo que el remitente organice la información que envía en tramas de datos (típicamente unos pocos cientos o miles de bytes) y transmitirlas secuencialmente. Si el servicio es confiable, el receptor informará sobre la correcta recepción de cada trama enviando su propia trama de confirmación.

Las redes que admiten *broadcast* tienen un problema adicional en la capa de datos: ¿cómo controlar el acceso a un canal de uso compartido? Una subcapa especial de la capa de datos, la subcapa MAC (*Medium Access Control*), lidia con este problema.

Capa de red.

Un problema clave de diseño es determinar cómo son encaminados los paquetes desde una fuente hacia un destino. Las rutas pueden estar basadas en tablas estáticas que rara vez cambian o, más a menudo, en tablas que pueden ser actualizadas automáticamente para evitar fallas de comunicación por componentes dañados, por ejemplo. El control del tráfico a ese nivel es función de la capa de red.

Si muchos paquetes están presentes en una subred al mismo tiempo, está latente la posibilidad de que se formen cuellos de botella. Manejar la congestión es responsabilidad también de la capa de red, en conjunto con capas superiores, al adaptar la carga de información que se envía hacia la propia red. En términos generales, la calidad del servicio proporcionada (retrasos, tiempo de transmisión, oscilaciones, etcétera) es también una labor de la capa de red.

Capa de transporte.

La función básica de la capa de transporte es aceptar datos de capas superiores, separarlos en unidades de información más pequeñas si es necesario, pasarlos hacia la capa de red y asegurarse de que todas las "piezas" se recibirán correctamente en el otro extremo de la transmisión. Además, todo esto debe hacerse de manera eficiente, de tal forma que se aísle a las capas superiores de cambios inesperados en el *hardware* con el paso del tiempo.

La capa de transporte también determinará qué tipo de servicio se le dará a la capa de sesión y, por último, a los usuarios finales de la red. El tipo más popular de conexión es un canal punto a punto libre de errores que entrega mensajes (o bytes) en el orden en que fueron enviados por el remitente. Sin embargo, existen otros tipos de servicios de transporte, como el de envío de paquetes aislados que no garantiza la entrega en el orden en el que fueron enviados y la difusión de mensajes hacia múltiples destinatarios. El tipo de servicio se determina cuando se establece la comunicación.

La capa de transporte es verdaderamente una capa de extremo a extremo; transporta datos por todo el camino desde la fuente hasta el destino. En otras palabras, un programa en la máquina remitente mantiene una conversación con un programa similar en la computadora destino, haciendo uso de las cabeceras de los mensajes y segmentos de control.

Capa de sesión.

La capa de sesión permite a los usuarios de diferentes máquinas establecer sesiones entre ellos. Las sesiones ofrecen varios servicios, incluyendo control de diálogo (haciendo un seguimiento de quién tiene el turno para transmitir), que previene a los participantes de intentar la misma operación crítica simultáneamente (esto es conocido como *token management*); y la sincronización, que genera puntos de control cuando las transmisiones son largas para que los nodos puedan retomar la conexión en caso de que ocurra una falla de comunicación.

Capa de presentación.

A diferencia de capas inferiores, que generalmente se preocupan por el flujo de bits, la capa de presentación lidia con la sintaxis y la semántica de la información transmitida. Con el fin de hacer posible la comunicación entre las computadoras con diferentes representaciones internas de la información, las estructuras de datos que se intercambian pueden ser definidas en un camino abstracto junto con una codificación estándar que va a ser utilizada en el medio. La capa de presentación maneja estas estructuras de datos y permite a otras de mayor nivel ser definidas e intercambiadas.

Capa de aplicación.

La capa de aplicación contiene una variedad de protocolos que son comúnmente necesitados por los usuarios. Un protocolo de aplicación ampliamente utilizado es HTTP (*HyperText Transfer Protocol*), que es la base para la *World Wide Web*. Cuando un navegador busca una página en Internet, envía la petición de la página al servidor que la contiene usando HTTP. El servidor típicamente responderá enviando la página solicitada.

Otros protocolos de aplicación son usados para la transferencia de archivos, correo electrónico y notificaciones de la red, por dar algunos ejemplos.

2.1.5.2 TCP/IP.

TCP/IP fue el primer modelo de referencia utilizado en "el abuelo" de todas las redes de computadoras de área amplia –ARPANET–, que fue una red de investigación patrocinada por el DoD (U. S. *Department of Defense*). Eventualmente conectaba cientos de universidades e instalaciones gubernamentales, usando líneas telefónicas alquiladas. Cuando las redes satelitales y de radio fueron agregadas tiempo después, los protocolos existentes tuvieron problemas para interconectarse entre ellos, así que fue necesaria una nueva arquitectura de referencia. Dicha arquitectura fue bautizada posteriormente como "Modelo de Referencia TCP/IP", siendo estos sus dos protocolos primarios.

Las capas que componen el modelo TCP/IP se explicarán después del diagrama comparativo con el modelo OSI disponible en la Figura 2.2.

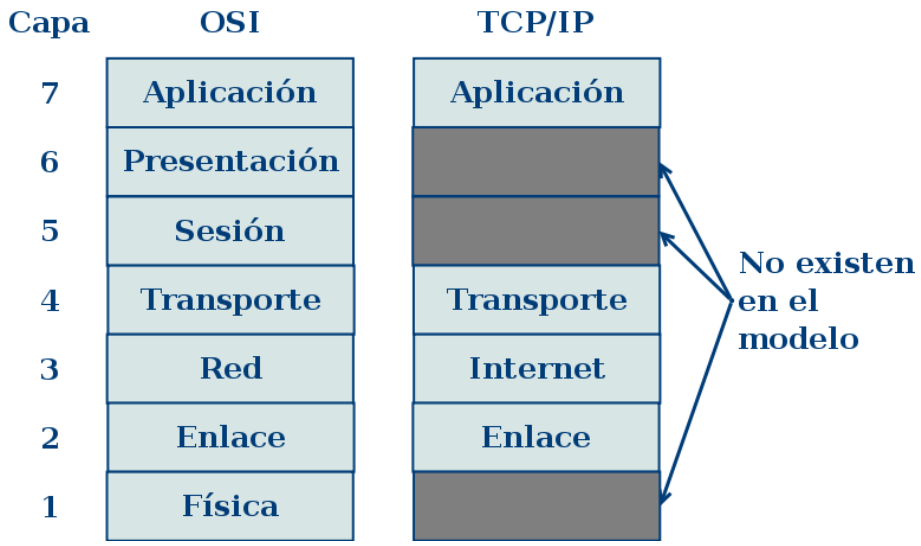

Figura 2.2. Comparación entre los modelos de referencia OSI y TCP/IP.

Capa de enlace.

Todos los requerimientos necesarios llevaron a la opción de tener una red de conmutación de paquetes basada en una capa sin conexión que funciona a través de diferentes redes. La capa más baja en el modelo –la capa de enlace– describe lo que los enlaces, tales como líneas seriales y Ethernet clásicas, deben hacer para satisfacer las necesidades de esta capa de acceso. No es realmente una capa como tal, en el sentido técnico del término, más bien es una interfaz entre los nodos y los enlaces de transmisión.

Capa de internet.

La capa de internet es el eje que mantiene a la arquitectura completa unida. Su trabajo es permitir a los nodos inyectar paquetes dentro de cualquier red y hacerlos viajar, independientemente de su destino (potencialmente una red diferente). Ellos pueden llegar incluso en un orden completamente distinto al que fueron enviados originalmente, en cuyo caso será trabajo de capas superiores reordenarlos, si una entrega ordenada es necesaria. Note que el término "internet" es usado aquí en un sentido genérico, a pesar de que esta capa está presente en la Internet.

La capa de internet define un formato de paquetes oficial y un protocolo llamado IP (*Internet Protocol*), más un protocolo compañero llamado ICMP (*Internet Control Message Protocol*) que ayuda principalmente en la detección de errores. El trabajo de la capa de internet es entregar paquetes IP a donde se supone que ellos se dirigen. El encaminamiento de paquetes es claramente la mayor tarea aquí, así como el manejo de la congestión (aunque IP por sí solo no ha demostrado ser capaz de evitarla).

Capa de transporte.

La capa superior a la capa de internet en el modelo TCP/IP es actualmente llamada capa de transporte. Está diseñada para permitir a los elementos que se comunican dentro de una red llevar a cabo una conversación, tal como en la capa de transporte del modelo OSI. Dos protocolos de transporte de punto a punto han sido definidos aquí. El primero de ellos, TCP (*Transmission Control Protocol*), es un confiable protocolo orientado a conexión que permite un flujo de bytes originados en una máquina para que sean entregados sin error en cualquier otra máquina en la Internet. Su trabajo es segmentar la transmisión de bytes entrante en mensajes discretos y pasar cada uno a la capa de internet. En el destino, el proceso de recepción TCP ensambla los mensajes recibidos en una corriente de salida. TCP también se ocupa del control de flujo para prevenir que un emisor que procesa muy rápido no inunde a un receptor con procesamiento lento, enviándole más mensajes de los que puede manejar.

El segundo protocolo en esta capa es UDP (*User Datagram Protocol*). Se trata de un protocolo poco fiable no orientado a conexión para aplicaciones que no requieran de la secuenciación o el control de flujo que ofrece TCP y que provean un mecanismo propio para lograrlo.

Capa de aplicación.

En el diseño del modelo TCP/IP no fue necesaria la presencia de las capas de sesión y de presentación. En lugar de eso, las aplicaciones simplemente incluyen cualquier función de sesión y presentación que ellas requieran. La experiencia con el modelo OSI ha probado que esto es correcto: dichas capas tienen poco uso en la mayoría de las aplicaciones.

En la cima del modelo TCP/IP está la capa de aplicación, que contiene todos los protocolos de alto nivel. Los primeros incluían terminales virtuales (TELNET), transferencia de archivos (FTP) y correo electrónico (SMTP). Varios más fueron incluidos con el paso de los años, como el DNS, HTTP y RTP.

La Figura 2.3 muestra algunos protocolos o servicios y su relación con las capas del modelo TCP/IP.

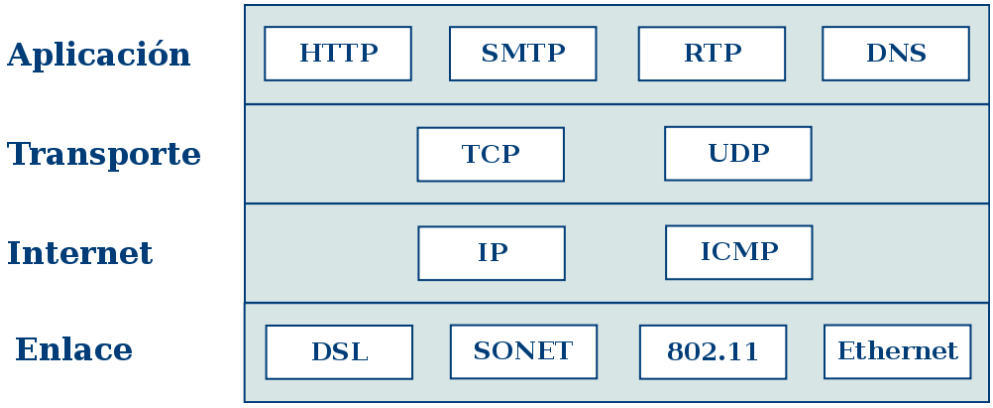

Figura 2.3. Protocolos en las capas del modelo de referencia TCP/IP.

2.1.6 Definición de protocolo.

Toca el turno de explicar un término muy importante dentro del argot de las redes de computadoras. La palabra "protocolo" aparecerá con frecuencia en lo que resta de este documento y su trascendencia es fundamental para lo que se pretende lograr.

Probablemente la manera más fácil de adquirir la noción de qué es un protocolo de redes de datos es considerando algunas analogías humanas, teniendo en cuenta que los seres humanos empleamos protocolos todo el tiempo. Consideremos la situación en la que alguien nos pregunta por la hora. El protocolo humano (o por lo menos la forma correcta de hacerlo) dice que primero se debe saludar a la persona para iniciar la comunicación con ella, tal vez con un "hola", para ejemplificarlo de la manera más coloquial. La típica respuesta a un "hola" es devolver el saludo con otro mensaje que diga "hola". Implícitamente, recibir una respuesta positiva es indicativo de que se puede hacer la pregunta sobre la hora del día. Una respuesta diferente al "Hola" inicial (como un "no me molestes" o cualquier otra no deseable) debe indicar una incapacidad para comunicarse, en cuyo caso el protocolo humano dicta que no podríamos preguntar por la hora del día. Note que en el protocolo humano existen mensajes específicos que enviamos y acciones específicas que realizamos en respuesta a las contestaciones recibidas u otros eventos probables (como no recibir una respuesta después de un cierto período de tiempo).

Claramente, los mensajes recibidos y transmitidos y las acciones tomadas cuando estos eventos ocurrieron, juegan un rol central en el protocolo humano. Si la gente sigue protocolos diferentes (por ejemplo, si la otra persona tiene costumbres diferentes o no habla el mismo idioma) los protocolos no podrán compartir información útil y ningún trabajo productivo será completado. La misma situación ocurre con los protocolos en las redes de datos: en ellas existen dos o más entidades ejecutando el mismo protocolo para completar una tarea.

De acuerdo a la analogía anterior, un protocolo de red es similar a un protocolo humano, exceptuando que las entidades que intercambian mensajes y toman acciones son componentes de *hardware* o *software* de un dispositivo (por ejemplo, una computadora, un *smartphone*, una *tablet*, un *router* o cualquier otro componente de una red). Toda la actividad en una red de datos que involucra a dos o más entidades remotas comunicándose es gobernada por al menos un protocolo. Por ejemplo, de manera muy general: los protocolos implementados en *hardware* dentro de dos computadoras físicamente interconectadas controlan el flujo de bits en el medio entre las dos tarjetas de interfaz de red; los protocolos de congestión en los sistemas controlan la tasa a la que son enviados los paquetes entre el transmisor y el receptor; los protocolos en los *routers* determinan qué trayectoria seguirá un paquete desde su origen hasta su destino.

Formalmente, podemos considerar como válida la siguiente explicación sobre las funciones de un protocolo de red:

"Un protocolo define el formato y el orden de los mensajes intercambiados entre dos o más elementos que se comunican, así como las acciones tomadas en la transmisión y/o recepción de un mensaje u otro evento."[9]

2.1.7 Puertos de transporte.

El intercambio de información en las redes de datos se hace posible mediante direcciones físicas (MAC) y direcciones lógicas (IP). Gracias a esos dos identificadores, un dispositivo tiene los elementos necesarios para localizar a otro y establecer comunicación con él. No obstante, conocer las direcciones de ambos no es suficiente para que exista un entendimiento, también es importante el dominio de los puertos en la capa de transporte.

En un Sistema Operativo, un puerto en la capa de transporte es una interfaz virtual imprescindible para comunicarse con un programa a través de una red. En el modelo OSI, la capa de transporte es donde se administra el uso y disponibilidad de los puertos, agregándolos al encabezado de los segmentos y posibilitando así el envío y el posterior ensamblaje de cada uno de ellos cuando llegan al dispositivo destino. Un puerto se identifica a través de un número para poder relacionarlo con la aplicación que lo utiliza en

[9] KUROSE, James. Keith, Ross. "Computer Networking: A Top-Down Approach. Featuring the Internet".

ese momento o decidir a qué programa entregará los datos recibidos. La asignación de puertos en el sistema permite a una computadora establecer simultáneamente diversas conexiones con nodos distintos: a pesar de que todos los segmentos van dirigidos hacia una misma dirección o provienen de la misma, se identifican con puertos diferentes.

El espacio destinado a la identificación del puerto tiene una longitud de 2 bytes, por lo tanto existen 65535 puertos disponibles (sin contar el 0) en un Sistema Operativo. Aunque teóricamente se puede disponer de cualquiera de ellos para ligarlo a un protocolo, la IANA (*Internet Assigned Numbers Authority*) creó tres categorías posibles:

- Puertos bien conocidos. Los inferiores a 1024 son puertos reservados para el Sistema Operativo y son usados por "protocolos bien conocidos". Tal es el caso de HTTP (puerto 80), HTTPS (443), SMTP (25), FTP (20 y 21), SSH (22) y Telnet (23); sólo por mencionar algunos.
- Puertos registrados. Los comprendidos entre 1024 y 49151 son denominados "registrados" y pueden ser usados por cualquier aplicación. Existe una lista pública en el sitio de la IANA donde se puede ver qué protocolo se relaciona a cada uno de ellos.
- Puertos dinámicos o privados. Están en el rango de 49152 a 65535 y normalmente se asignan de forma dinámica a las aplicaciones de clientes que quieren establecer una conexión.

2.2 Firewall.

La habilidad de conectar cualquier computadora con otra computadora, dondequiera que estén, es un arma de doble filo. Para las personas que tienen una red casera, deambular por la Internet puede ser muy divertido. Para los administradores corporativos de seguridad resulta ser una pesadilla. La mayoría de las compañías tienen grandes cantidades de información confidencial disponible en línea (secretos y estrategias de mercado, planes de desarrollo de nuevos productos, análisis financieros, etcétera). Revelar esa información a un competidor suele tener serias consecuencias.

Adicionalmente al peligro de la fuga de información al exterior, existe el mismo problema en sentido opuesto. En particular, virus, gusanos y otras pestes digitales pueden vulnerar la seguridad, destruyendo información valiosa y empleando gran cantidad de tiempo de los administradores en arreglar el lío que producen. A menudo estos problemas son traídos por empleados descuidados que buscan entretenerse en algún nuevo juego.

Consecuentemente, se necesitan mecanismos para mantener los "bits buenos" dentro y los "bits malos" fuera. Un método viable es utilizar IPsec, un conjunto de protocolos de seguridad que se verá posteriormente con mayor detenimiento. Pero a pesar de ser muy útil, este enfoque sólo protege la información en tránsito entre dos sitios seguros sin hacer algo

por mantener a los intrusos y a las pestes digitales lejos de la LAN de la compañía. Para lograr tal objetivo son necesarios los *firewalls*.

Un *firewall* es una combinación de *hardware* y *software* que aísla la red interna de una organización de las amenazas que abundan en la Internet, permitiendo el flujo de algunos paquetes y bloqueando o dándole un trato especial a los restantes. De manera más concisa, el *firewall* actúa como un filtro de paquetes inspeccionando todos y cada uno de ellos tanto a la entrada como a la salida del propio dispositivo. Los paquetes cumplen con cierto criterio descrito en las reglas formuladas por el administrador de la red para que puedan ser procesados y reenviados con normalidad. Aquellos que no pasen la prueba serán bloqueados o desechados sin contemplaciones. Ver Figura 2.4.

El criterio de filtrado está dado típicamente como reglas o tablas que listan fuentes y destinos que son aceptables, fuentes y destinos que deben ser bloqueados y reglas por defecto que determinan qué hacer con los paquetes que no coincidan con ninguna de las reglas específicas. En el caso muy común de una conexión TCP/IP, una fuente o un destino puede consistir en una dirección IP, una dirección MAC, un protocolo, un puerto o la combinación de ellos. Los puertos indican cuál es el servicio deseado.

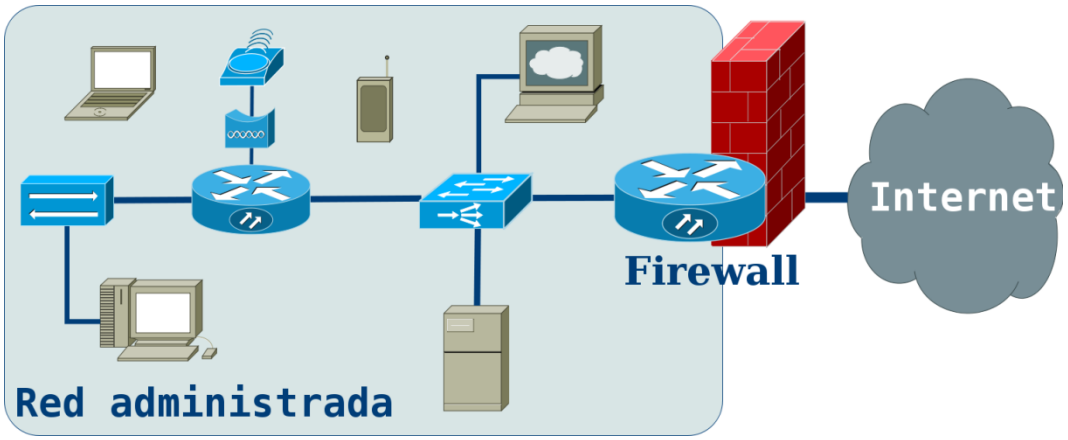

Figura 2.4. Localización más común de un *firewall* dentro de una red.

Un *firewall* tiene al menos tres rasgos importantes que lo caracterizan:

- Todo el tráfico de fuera hacia dentro, y viceversa, pasa a través del *firewall*.
- Sólo el tráfico autorizado, tal como se definió en la política de seguridad local, tendrá libre tránsito.
- El *firewall* por sí mismo es inmune a la vulneración. Al ser un elemento que está conectado a la red, si el *firewall* no está diseñado o instalado apropiadamente puede verse comprometido, en cuyo caso sólo da un falso sentido de seguridad.

2.2.1 DMZ (*DeMilitarized Zone*).

Algunos puertos no pueden ser fácilmente bloqueados. La dificultad radica en que los administradores de la red necesitan seguridad pero no pueden cortar de tajo la comunicación con el mundo exterior. Aquí es donde un concepto como el de zona desmilitarizada o DMZ (por sus siglas en inglés: *DeMilitarized Zone*) se vuelve muy práctico. La DMZ es la parte de la red de la compañía que se encuentra fuera del perímetro de seguridad. Cualquier cosa podría ir ahí si se permite. Colocando una máquina como un servidor web en la DMZ, las computadoras pueden consultarlo a través de una página publicada en Internet.

Podemos definir la DMZ como una red, o parte de ella, separada de otras partes de la red corporativa por medio de un *firewall* que permite que sólo entre o salga cierta clase de tráfico. El objetivo principal es que todo el tráfico externo (el de Internet, en este caso) se comunique sólo con la DMZ.

La comunicación entre los elementos que pertenecen a la DMZ y la red interna debe ser, por diseño, estrictamente restringida o nula, si es posible. De esa forma se previenen ataques hacia toda la red corporativa en caso de que algún intruso logre tomar el control de los nodos expuestos en la DMZ.

Un diagrama típico del lugar que ocupa la DMZ en la red de una organización es el de la Figura 2.5.

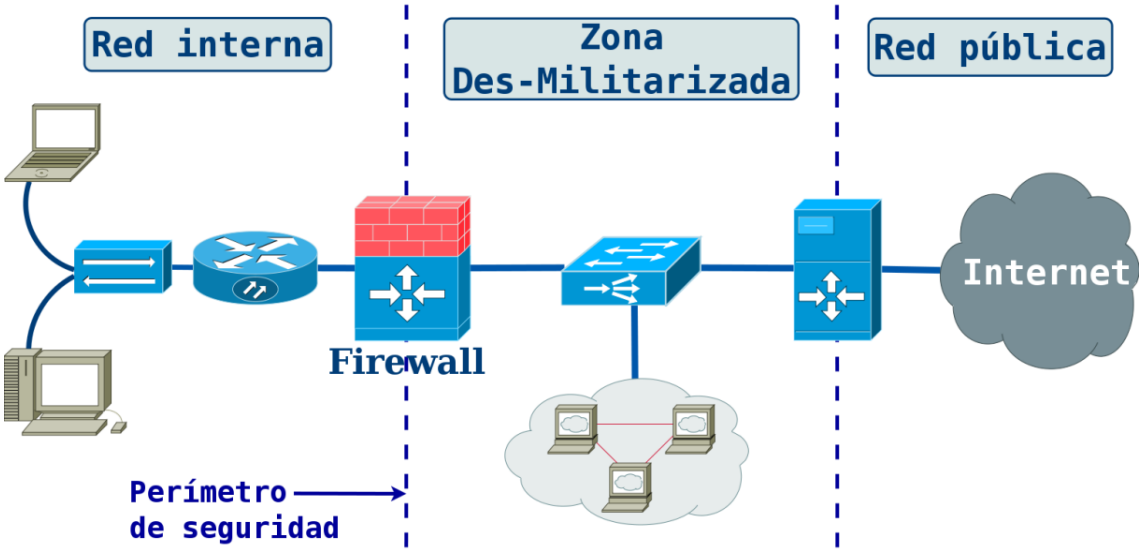

Figura 2.5. Ubicación de la DMZ en una red.

2.3 VPN (*Virtual Private Network*).

Toca el turno de abordar un tópico que se ha vuelto imprescindible en la supervivencia de las compañías con presencia en múltiples ciudades o que dentro de sus necesidades está dar acceso remoto bajo demanda a los empleados; e incluso interconectarse con otras redes de clientes y proveedores cuyas instalaciones son muy distantes.

En otros tiempos, antes de que las redes públicas de datos existieran, era común que algunas empresas rentaran líneas desde la compañía telefónica hacia sus sucursales. Actualmente ésta práctica se ha ido quedando atrás. A una red construida a partir de computadoras propias y conectadas mediante líneas telefónicas alquiladas se le llama red privada.

Las redes privadas funcionan bien y son muy seguras. Si las únicas líneas disponibles son las alquiladas, el tráfico no podrá entrar ni salir de la compañía, de tal manera que los intrusos deberán conectarse físicamente cortando las líneas, que no es algo discreto y sencillo de hacer. El problema con las redes privadas es que al contratar enlaces dedicados nada es barato; dependiendo de la distancia y la capacidad del medio, las rentas van desde cientos hasta miles de dólares por mes.

Con la aparición de las redes de datos y la Internet, muchas compañías buscaron mover su tráfico de datos hacia las redes públicas, pero sin gozar de la seguridad que otorgan las redes privadas. Esta demanda pronto llevó a la invención de las redes privadas virtuales (VPN, por sus siglas en inglés), que son enlaces virtuales creados sobre redes públicas que ofrecen prácticamente todas las propiedades necesarias de las redes privadas.

Algo muy popular es construir VPNs directamente por Internet. El diseño más común es equipar a cada sucursal con un *firewall* y crear túneles a través de Internet entre todos los pares de oficinas, como se ilustra en la Figura 2.6(a). La flexibilidad es mucho mayor de la que se tiene con enlaces dedicados; visto desde la perspectiva de los usuarios de una VPN, la topología luce tal como en una red privada. Así lo muestra la Figura 2.6(b).

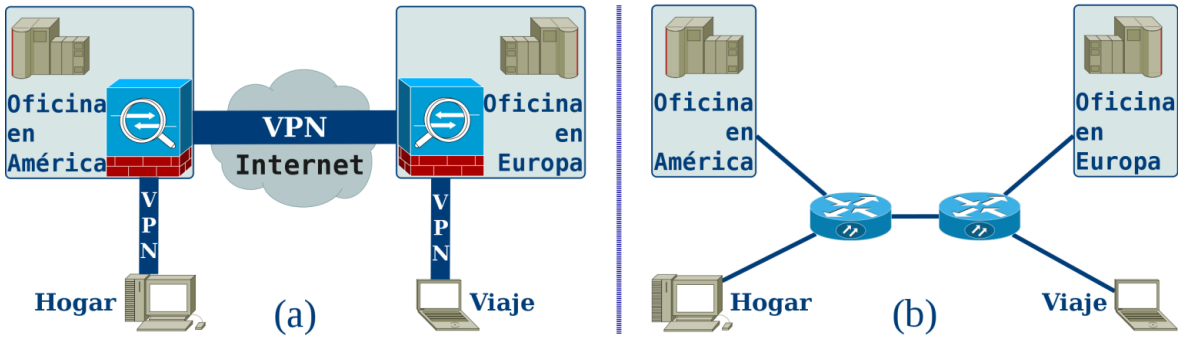

Figura 2.6. (a) Sitios geográficamente distantes conectados por VPN. (b) Topología equivalente de una red conectada por VPN.

Cuando el sistema se pone en funcionamiento, cada par de *firewalls* debe negociar los parámetros de su SA (*Security Association*), incluyendo los servicios, modos, algoritmos y llaves. Si los protocolos de IPsec son utilizados para crear los túneles, es posible incluir todo el tráfico entre ambas sucursales en una sola SA que provee control de integridad, discreción y una considerable inmunidad al análisis de tráfico.

Una vez que las SA se han establecido, el tráfico puede comenzar a fluir. Para un *router* dentro de Internet, un paquete viajando por un túnel VPN es un paquete ordinario. La única propiedad inusual en él es la presencia de un encabezado IPsec después del encabezado IP, pero debido a que los encabezados extra no tienen efecto en el proceso de reenvío de paquetes, los *routers* no lo toman en cuenta.

Una ventaja clave de una VPN es ser completamente transparente para todo el *software* del usuario. Los *firewalls* son los que establecen y administran las SAs y la única persona que está consciente de esta situación es el administrador del sistema o de la red. Él es quien debe configurar y mantener los elementos de seguridad, mientras que todos los demás usuarios tendrán la misma precepción que si tuvieran una línea dedicada para su red privada.

2.3.1 IPsec.

IPsec es una extensión de protocolo IP que proporciona seguridad a éste y a los protocolos de capas superiores. Fue desarrollado para el nuevo estándar IPv6 y después fue portado a IPv4. La arquitectura IPsec se describe en el RFC2401. Los siguientes párrafos dan una pequeña introducción a IPsec.

IPsec emplea dos protocolos diferentes –AH y ESP– para asegurar la autenticación, integridad y confidencialidad de la comunicación. Puede proteger el datagrama IP completo o sólo los protocolos de capas superiores. Estos modos se denominan, respectivamente, modo túnel y modo transporte. En modo túnel el datagrama IP se encapsula completamente dentro de uno nuevo que emplea el protocolo IPsec. En modo transporte IPsec sólo maneja la carga del datagrama IP, insertándose la cabecera IPsec entre la cabecera IP original y la cabecera del protocolo de capas superiores. Gráficamente se puede entender mejor en la Figura 2.7.

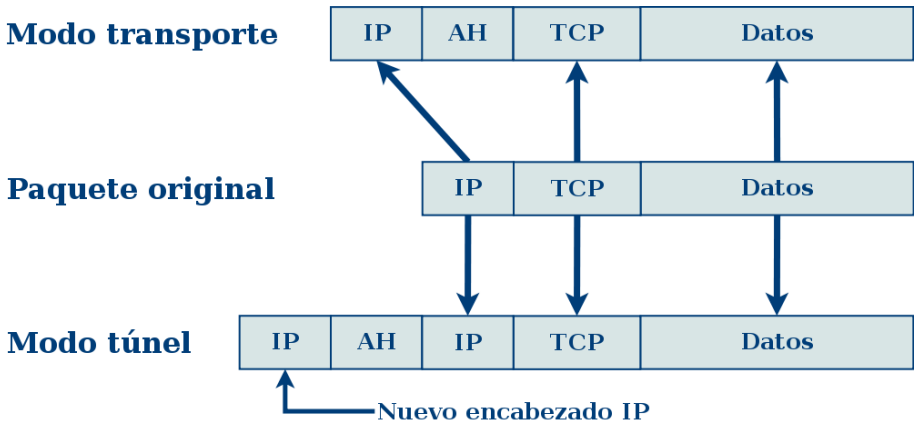

Figura 2.7. Comparación entre las cabeceras de los modos de IPsec.

Para proteger la integridad de los datagramas IP, los protocolos IPsec utilizan códigos de autenticación de mensaje basados en resúmenes (HMAC: *Hash Message Authentication Codes*). Para la generación de los HMAC, los protocolos usan algoritmos como MD5 y SHA para calcular un resumen basado en una clave secreta y en los contenidos del datagrama IP. El HMAC se incluye en la cabecera del protocolo IPsec y el receptor del paquete puede comprobar el HMAC si tiene acceso a la clave secreta.

La confidencialidad de los mensajes se garantiza con el uso de algoritmos estándar de cifrado simétrico. El estándar IPsec exige la implementación de NULL y DES. En la actualidad se suelen emplear algoritmos más fuertes: 3DES, AES y *Blowfish*.

Para estar prevenidos contra ataques por denegación de servicio (DoS: *Denial of Service*), los protocolos IPsec emplean ventanas deslizantes. Cada paquete recibe un número de secuencia y sólo se acepta su recepción si el número de paquete se encuentra dentro de la ventana o es posterior. Los paquetes anteriores son descartados inmediatamente. Ésta es una medida de protección eficaz contra ataques por repetición de mensajes en los que el atacante almacena los paquetes originales y los reproduce posteriormente.

Para que los participantes de una comunicación puedan encapsular los paquetes IPsec y hacer el proceso inverso, se necesitan mecanismos para almacenar las claves secretas, algoritmos y direcciones IP involucradas en la comunicación. Todos estos parámetros se almacenan en SAs. Las asociaciones de seguridad, a su vez, se almacenan en SAD (*Security Association Databases*).

Cada asociación de seguridad define los siguientes parámetros:

- Dirección IP origen y destino de la cabecera IPsec resultante. Estas son las direcciones IP de los participantes de la comunicación IPsec que protegen los paquetes.

- Protocolo IPsec (AH o ESP). A veces, se permite compresión –IPCOMP–.
- El algoritmo y clave secreta empleados por el protocolo IPsec.
- Índice de parámetro de seguridad (SPI: *Security Parameter Index*). Es un número de 32 bits que identifica la asociación de seguridad.

Algunas implementaciones de la base de datos de asociaciones de seguridad –SAD– permiten almacenar más parámetros:

- Modo IPsec (túnel o transporte).
- Tamaño de la ventana deslizante para protegerse de ataques por repetición.
- Tiempo de vida de una asociación de seguridad.

En una asociación de seguridad se definen las direcciones IP de origen y destino de la comunicación. Por ello, mediante una única SA sólo se puede proteger un sentido del tráfico en una comunicación IPsec *full duplex*. Para proteger ambos sentidos de la comunicación, IPsec requiere dos SA unidireccionales.

Las asociaciones de seguridad sólo especifican cómo se supone que IPsec protegerá el tráfico. Para definir qué tráfico proteger y cuándo hacerlo, se necesita información adicional. Esta información se almacena en la política de seguridad (SP: *Security Policy*), que a su vez se almacena en la base de datos de políticas de seguridad (SPD: *Security Policy Database*).

Una política de seguridad suele especificar los siguientes parámetros:

- Direcciones IP de origen y destino de los paquetes por proteger. En modo transporte, éstas serán las mismas direcciones que en la SA; en modo túnel pueden ser distintas.
- Protocolos y puertos a proteger. Algunas implementaciones no permiten la definición de protocolos específicos a proteger. En ese caso, se protege todo el tráfico entre las direcciones IP indicadas.
- La asociación de seguridad a emplear para proteger los paquetes.

La configuración manual de la asociación de seguridad es proclive a errores y no es muy segura. Las claves secretas y algoritmos de cifrado deben ser compartidas entre todos los participantes de la VPN. Uno de los problemas críticos a los que se enfrenta el administrador de sistemas es el intercambio de claves: ¿cómo intercambiar claves simétricas cuando aún no se ha establecido ningún tipo de cifrado?

Para resolver este problema se desarrolló el protocolo de intercambio de claves por Internet (IKE: *Internet Key Exchange*). Este protocolo autentica a los participantes en una primera fase. En una segunda fase se negocian las asociaciones de seguridad y se escogen las claves secretas simétricas a través de un intercambio de claves de tipo Diffie Hellmann. El protocolo IKE se ocupa incluso de renovar periódicamente las claves para asegurar su confidencialidad.

Protocolos de IPsec.

La familia de protocolos IPsec está formada por dos protocolos: AH y ESP. Ambos son protocolos IP independientes. AH es el protocolo IP 51 y ESP el protocolo IP 50.

AH (*Authentication Header*).

El protocolo AH protege la integridad del datagrama IP. Para conseguirlo, calcula una HMAC basada en la clave secreta, el contenido del paquete y las partes inmutables de la cabecera IP (como son las direcciones IP). Tras esto, añade la cabecera AH al paquete.

Como el protocolo AH protege la cabecera IP incluyendo sus partes inmutables, no permite NAT (*Network Address Translation*). NAT reemplaza una dirección IP de la cabecera IP por una dirección IP diferente. Tras el intercambio, la HMAC ya no es válida. La extensión de IPsec, NAT-T (o NAT-Transversal), implementa métodos que evitan esta restricción.

ESP (*Encapsulating Security Payload*).

El protocolo ESP puede asegurar la integridad del paquete empleando una HMAC y la confidencialidad con cifrado. La cabecera ESP se genera y se añade al paquete tras cifrarlo y calcular su HMAC.

El uso de NAT, por lo tanto, no rompe el protocolo ESP. Sin embargo, en la mayoría de los casos NAT aún no es compatible en combinación con IPsec. Aquí, NAT-T también ofrece una solución para este problema encapsulando los paquetes ESP dentro de paquetes UDP.

IKE (*Internet Key Exchange*).

El protocolo IKE resuelve el problema más importante del establecimiento de comunicaciones seguras: la autenticación de los participantes y el intercambio de claves simétricas. Tras ello, crea las SA y rellena la SAD. El protocolo IKE emplea el puerto 500 de UDP para su comunicación.

El protocolo IKE funciona en dos fases. La primera fase establece una ISAKMP SA (*Internet Security Association Key Management Protocol - Security Association*). En la segunda fase, la ISAKMP SA se emplea para negociar y establecer las SA de IPsec.

La autenticación de los participantes en la primera fase suele basarse en claves compartidas con antelación (PSK: *Pre-Shared Key*), con claves RSA (Rivest, Shamir *and* Adleman) y con certificados X.509.

La primera fase suele soportar dos modos distintos: modo principal y modo agresivo. Ambos modos autentican al participante en la comunicación y establecen una ISAKMP SA, pero el modo agresivo sólo usa la mitad de mensajes para alcanzar su objetivo. El modo

agresivo, sin embargo, tiene sus desventajas, ya que no soporta la protección de identidades y es susceptible a un ataque tipo "*man in the middle*" (por escucha y repetición de mensajes en un nodo intermedio) si se emplea junto con claves pre-compartidas. El modo agresivo transmite la identidad del cliente en claro y, por lo tanto, los participantes de la comunicación se conocen antes de que la autenticación se lleve a cabo. Así, se pueden emplear distintas PSK con distintos participantes.

En la segunda fase, el protocolo IKE intercambia propuestas de asociaciones de seguridad y las negocia basándose en la ISAKMP SA. La ISAKMP SA proporciona autenticación para protegerse de ataques "*man in the middle*".

Normalmente, los participantes de la comunicación sólo negocian una ISAKMP SA, que se emplea para establecer varias (al menos dos) IPsec SA unidireccionales.

2.4 GRE (*Generic Routing Encapsulation*).

Es un protocolo de túnel desarrollado por Cisco Systems que encapsula una amplia variedad de protocolos pertenecientes a la capa de internet TCP/IP dentro de un enlace virtual punto a punto que fluye sobre una interconexión de redes IP. Este protocolo provee un enfoque genérico sencillo para transportar paquetes de un protocolo sobre otro protocolo por medio de encapsulamiento.

GRE se lista dentro de la pila de protocolos IP y se identifica con el número 47. Su función es encapsular información o carga útil (*payload*), que es un paquete interior que necesita ser entregado a una red destino dentro de un paquete IP exterior. Los dispositivos finales entre los que se establece el enlace virtual, a menudo llamados *endpoints*, envían información a través del túnel GRE mediante el enrutamiento de paquetes encapsulados en donde intervienen redes IP. Los demás *routers* IP que están presentes en el trayecto no analizan la *payload*, ellos sólo interpretan el paquete IP externo para poder reenviarlo a su destino. Una vez que llega al *endpoint*, el encapsulamiento GRE es removido y la información se entrega al destinatario final.

Los túneles modo GRE son una buena aproximación al concepto de redes privadas virtuales, aunque no se consideran como herramienta útil para establecerlas debido que adolecen de las características de seguridad necesarias. Un túnel GRE es incapaz de garantizar la autenticidad del remitente ni la integridad del mensaje. Tampoco garantiza la confidencialidad de la información que está encapsulada en el paquete IP y los paquetes no están protegidos contra posibles reenvíos, por lo que el sistema es propenso a sufrir ataques de tipo DoS.

Las soluciones que utilizan GRE frecuentemente se diseñan para resolver problemas de enrutamiento a través de redes con listas de control de acceso muy restrictivas. La seguridad, si es requerida, se proporciona con herramientas y medios adicionales. Vea la situación mostrada en la Figura 2.8, que ejemplifica una solución de este tipo.

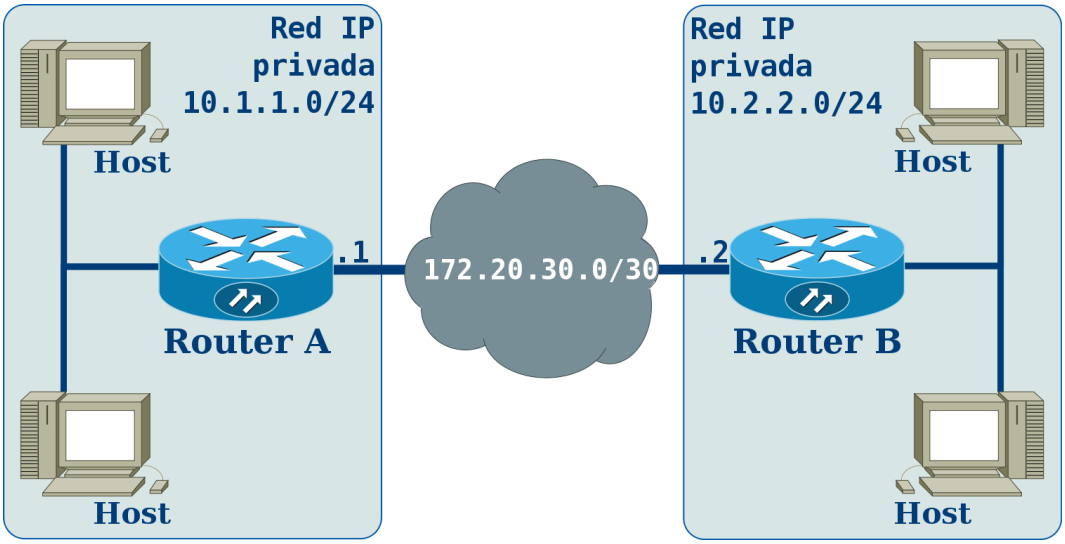

Figura 2.8. Esquema de dos redes potencialmente alcanzables con un túnel GRE.

De acuerdo al ejemplo de la Figura 2.8, los paquetes que salen desde una red IP privada 10.1.1.0/24 con destino a otra red IP privada con direccionamiento 10.2.2.0/24, son encapsulados por un *router* A y reenviados hacia el *router* B por medio de la red 172.20.30.0/30. El *router* B extrae la *payload* encapsulada en el paquete y la manda hacia el *host* destino, que es el servidor o usuario final.

2.5 *Routing*.

El rol de la capa de red del modelo OSI es sencillo en términos generales, pero si no se comprenden bien los procesos que hay detrás de su operación, el funcionamiento puede llegar a ser muy confuso para el usuario. Mover paquetes de un lado a otro implica dos funciones fundamentales en la capa de red:

- *Forwarding*. Cuando un paquete es recibido por un *router* en su interfaz de red, dicho *router* debe ser capaz de trasladarlo hacia la interfaz de salida apropiada, no sin antes haber analizado las cabeceras del paquete, determinar si es válido o no y cuál es el destino del mismo. El procesamiento de las unidades de información en capa 3 OSI es casi idéntico tanto en un *router* como en un *host*. La diferencia radica en que los *routers* tienen la función "IP *forwarding*" habilitada y los *hosts* no. Tener *forwarding* inhabilitado significa que los *hosts* sólo pueden transmitir un paquete IP si la información útil que lleva el paquete fue generada localmente (por él mismo). Un *host* nunca retransmitirá un paquete que haya recibido de algún otro elemento de la red.

- *Routing*. La capa de red es la encargada de determinar el trayecto que seguirán los paquetes para que fluyan desde su remitente hasta su receptor. Las decisiones de enrutamiento que tome un dispositivo estarán basadas en la tabla de *routing*, que son registros almacenados en memoria y que pueden ser manipulados manualmente por el administrador del sistema o dinámicamente con ayuda de un protocolo de enrutamiento. Dondequiera que un *host* o un *router* necesite transmitir información, realizará primero la búsqueda de una entrada en su tabla de *routing* que incluya el destino al que va dirigido el paquete. El resultado arrojará la dirección IP del siguiente dispositivo al que debe "saltar" el paquete o el nombre de la interfaz por la cual debe (re)enviarse.

Las tablas de *routing* pueden tener diferentes apariencias, dependiendo de los sistemas operativos del nodo en cuestión. Empero, como mínimo deben tener dos columnas: una que contiene las direcciones IP de destino y otra que especifica cómo o hacia dónde dirigir el paquete. La Tabla 2.1 ejemplifica una manera simple de concebir la tabla de *routing*.

Dirección destino	Siguiente salto
Prefijo de red o Dirección IP de un nodo o Dirección de *loopback* o Ruta por defecto	Dirección IP del *router* de siguiente salto o Nombre de una interfaz de red

Tabla 2.1. Elementos básicos en una tabla de routing.

Las direcciones de destino en la tabla pueden ser prefijos de red, la dirección IP de un *host* en particular, una dirección de *loopback* o una ruta por defecto (casi siempre indicada con 0.0.0.0). Recordemos que las rutas por defecto están ligadas a la existencia de un *default gateway* –también llamado puerta de enlace predeterminada– y es utilizado para reenviar todos los paquetes de los que no se encontró una entrada específica que los incluya. La columna "Siguiente salto" de la Tabla 2.1 contiene la dirección IP del siguiente *router* por el que debe pasar el paquete o, en su defecto, el nombre de la interfaz de red por la que debe enviarse en caso de que se pueda entregar directamente a su destino. Las tablas de *routing* en los sistemas operativos actuales suelen tener columnas adicionales. Por ejemplo, en Linux siempre contiene la dirección IP del siguiente salto y el nombre de la interfaz por la cual puede ser alcanzada.

Las tablas de *routing* en los *routers* pueden llegar a tener hasta varias miles de entradas y son generalmente puestas ahí por protocolos de enrutamiento dinámico. Los *hosts*, por

otro lado, requieren sólo unas cuantas rutas presentes en su tabla, que además pueden colocarse de manera estática y manual o mediante protocolos de red como DHCP.

La Figura 2.9 es un diagrama de bloques muy interesante que muestra cuál es el procedimiento general que se lleva a cabo en la capa de red.

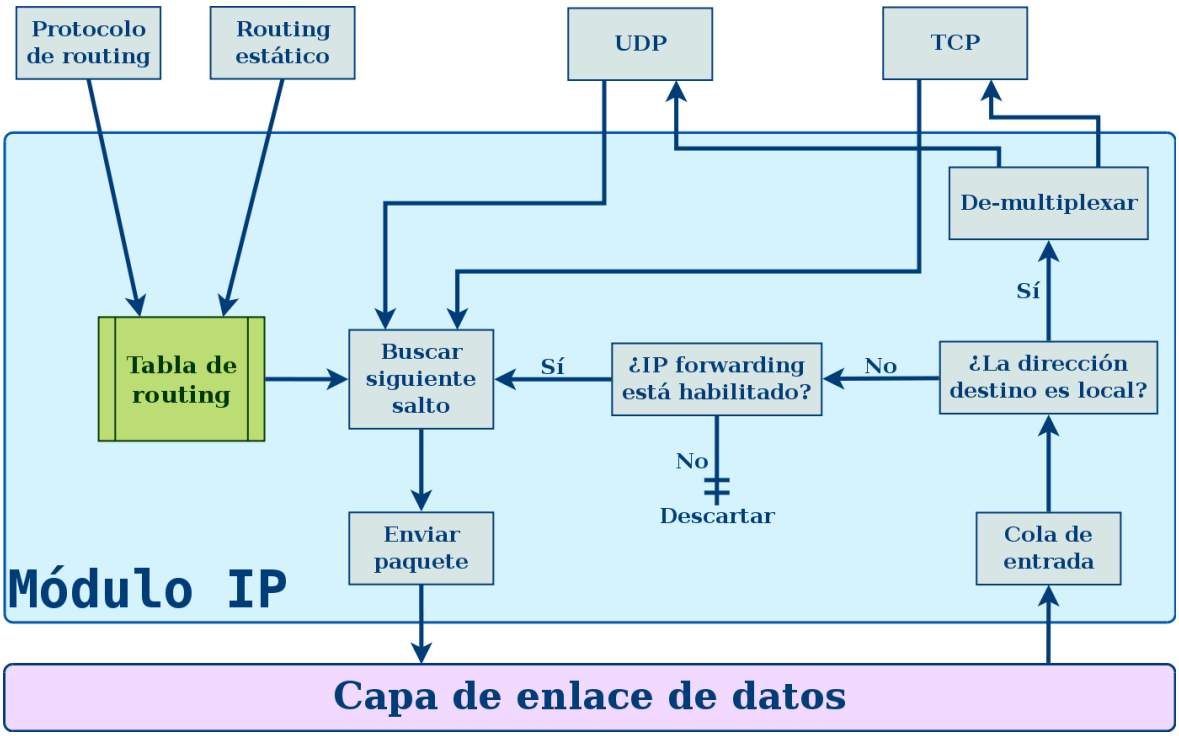

Figura 2.9. Procesamiento de un paquete en la capa 3 OSI.

Analicemos la Figura 2.9: cuando un paquete es recibido, pasando antes por la capa OSI de enlace de datos, el módulo IP verifica si el sistema local es el destino que viene en la cabecera. Si es así, el paquete es de-multiplexado y pasado hacia la capa superior inmediata para que protocolos tales como UDP y TCP se encarguen de definir las condiciones en las que será transportado el paquete. Si, por el contrario, el sistema local no es el destino, entonces procede una verificación para saber si IP *forwarding* está habilitado. De ser así, quiere decir que el sistema es un *router* y entonces se hará una búsqueda en la tabla de *routing* en un intento por saber hacia dónde deberá reenviarse la información. Si la función de reenvío no está disponible, entonces el sistema es un *host* y el paquete será descartado.

2.5.1 *Routing* dinámico.

Dado que el tamaño de la tabla de *routing* en un dispositivo puede llegar a ser gigantesco, sobre todo en *routers* de Internet o de algunas empresas considerablemente grandes, sería casi imposible que los administradores mantuvieran coherente y sana la comunicación entre sus dispositivos si tienen que hacer modificaciones manuales. Cualquier falla, cualquier desconexión o cambio en la topología de la red desataría una serie de inconsistencias en cadena que tardarían horas en resolver, afectando masivamente a los empleados de la compañía y, peor aún, a sus clientes.

Ante la necesidad de facilitar a los administradores un control adecuado en las configuraciones de todos sus equipos, se diseñaron útiles algoritmos que dieron origen a protocolos robustos para el intercambio y mantenimiento de las tablas de *routing*. Estos protocolos han ido evolucionando y dando origen a algunos nuevos y mejorados, ofreciendo la posibilidad de sostener un gran número de rutas que se agregan y se eliminan automáticamente en función de los incidentes que suceden en la red, además de incorporar seguridad para evitar compartir información con *routers* no deseados. Un buen protocolo de *routing* dinámico ofrece todo lo anterior y varias características más con el mínimo de configuración posible.

Todos los protocolos de *routing* dinámico tienen el mismo propósito: aprender acerca de las redes remotas y adaptarse rápidamente cuando existe un cambio en la topología. El método para lograr ese objetivo depende del algoritmo que utilice y de las características operacionales del protocolo. En general, las rutinas de un protocolo de *routing* dinámico se pueden englobar de la siguiente forma:

1) El *router* envía y recibe mensajes de *routing* en sus interfaces habilitadas para descubrir dispositivos vecinos.
2) El *router* comparte información propia y de configuración para sincronizarse con otros *routers* que utilizan el mismo protocolo.
3) Los *routers* intercambian información para aprender sobre redes remotas y guardan rutas para llegar a ellas.
4) Cuando un *router* detecta un cambio en la topología, el protocolo puede anunciar este cambio a sus vecinos para que modifiquen su tabla de *routing*.

Existen diferentes protocolos de *routing* disponibles que además son multiplataforma, lo que quiere decir que pueden utilizarse independientemente del Sistema Operativo que tengan los dispositivos.

La Tabla 2.2 nos da un comparativo entre *routing* estático y *routing* dinámico. En ella se observan sus ventajas y desventajas más sobresalientes.

Característica	*Routing* dinámico	*Routing* estático
Complejidad de configuración	Generalmente independiente del tamaño de la red	Incrementa con el tamaño de la red
Conocimientos necesarios del administrador	Requiere conocimientos de *routing* avanzados	Routing básico. No se requiere conocimiento extra
Cambios en la topología	Se adapta automáticamente a los cambios en la topología	Necesita la intervención del administrador
Escalabilidad	Adecuado para topologías simples y complejas	Adecuado sólo para topologías simples
Seguridad	Menos seguro, debido al constante intercambio de mensajes	Más seguro. No hay interacción con *routers* vecinos
Uso de recursos	Demanda mayor trabajo de "CPU", memoria y ancho de banda	Necesidad despreciable de memoria y procesamiento. No necesita recursos extra
Previsibilidad	El enrutamiento depende de la topología actual	Los destinos de enrutamiento son siempre los mismos

Tabla 2.2. Características comparables entre routing dinámico y estático.

No será necesario hacer mención de los diversos protocolos que existen en la actualidad y mucho menos las diferentes clasificaciones en las que se ordenan. Únicamente veremos la división de protocolos en IGP y EGP; y el caso muy particular de BGP en ambos escenarios.

Un AS (*Autonomous System*), también conocido como "dominio de *routing*", es un conjunto de *routers* bajo una administración en común. Los ejemplos típicos son la red interna de una compañía o una red de ISPs. Debido a que Internet está basado en el concepto de Sistema Autónomo, dos tipos de protocolo de *routing* son requeridos: interiores y exteriores. Esta clase de protocolos formalmente son:

- IGP (*Interior Gateway Protocol*). Son empleados para hacer *routing* dentro de un AS, tarea conocida en inglés como "*intra-autonomous system routing*". Controla el intercambio de rutas entre dispositivos de aquellas redes que están bajo el dominio de una sola organización y suele componerse de varias redes individuales pertenecientes a compañías, escuelas y otras instituciones.

- EGP (*Exterior Gateway Protocol*). Usados para *routing* entre distintos AS, también llamado "*inter-autonomous system routing*". Está diseñado para el intercambio de rutas entre dominios que están bajo el control de diferentes administraciones. Actualmente BGP es el único EGP viable y es el protocolo de *routing* utilizado en Internet; es capaz de utilizar diferentes atributos para evaluar las rutas. Es típicamente socorrido por los ISP para poder comunicarse entre ellos y, a veces, entre una compañía y un ISP. A este nivel, a menudo hay cuestiones más importantes que simplemente elegir la trayectoria más rápida.

El Sistema Autónomo de una organización, se identifica hacia el exterior con un número de longitud de dos bytes, lo cual ofrece la posibilidad de tener 65536 identificadores diferentes. Por ser una cantidad finita de opciones, la IANA se ocupa de controlar el uso de dichos identificadores para que su explotación sea la correcta. El rango privado va de 64512 a 65534 y es el destinado para uso libre interno. El resto de bloques están reservados o asignados para otro fin, incluidos el 0 y el 65535.

Convergencia.

Una característica importante de los protocolos de *routing* es: qué tan rápido convergen los elementos que los utilizan cuando se presenta un cambio en la topología.

La convergencia ocurre cuando las tablas de *routing* de todos los *routers* llegan a un estado de consistencia. Se dice que la red converge cuando todos los *routers* tienen una completa y precisa información sobre ella. El tiempo de convergencia es el tiempo que toma a los *routers* compartir información, calcular las mejores trayectorias y actualizar sus tablas de *routing*. Una red no es completamente operable mientras no converja, por eso resulta lógico pensar que la mayoría de las redes requieren de tiempos de convergencia muy cortos.

La convergencia es tanto colaborativa como independiente. Los *routers* comparten información con los demás, pero el impacto de los cambios en la topología debe ser calculado independientemente para sus propias rutas.

Las propiedades de la convergencia incluyen la velocidad de propagación de la información de *routing* y el cálculo de las trayectorias óptimas. Los protocolos de *routing* pueden ser evaluados basándose en su velocidad de convergencia: mientras más rápida sea la convergencia, mejor es el protocolo de *routing*.

Métricas.

La métrica es un método para medir y comparar rutas. Los protocolos de *routing* usan métricas para determinar qué ruta representa la mejor trayectoria.

Existen casos donde el protocolo de *routing* aprende más de una ruta para llegar al mismo destino. Para seleccionar la mejor de ellas, el algoritmo debe ser capaz de evaluar y diferenciar entre las trayectorias disponibles. Tal propósito se logra utilizando la métrica. Una métrica es un valor usado por los protocolos de *routing* al momento de calcular los costos para alcanzar redes remotas y determinar qué trayectoria es preferible hacia la misma red.

Cada protocolo tiene su propio algoritmo para calcular sus métricas. En consecuencia, todos ellos usan diferentes clases de métrica. Por lo tanto, la métrica obtenida por un protocolo de *routing* no es comparable con la registrada por otro.

Los algoritmos utilizados para calcular las métricas de los protocolos de *routing* IP pueden tomar en cuenta los siguientes parámetros:

- Número de saltos. Una métrica simple que cuenta el número de *routers* por los que el paquete debe cruzar.
- Ancho de banda. Influye en la selección de la trayectoria, prefiriendo aquella con el ancho de banda más grande.
- Carga. Considera la cantidad de tráfico presente en el enlace.
- Retraso. Toma en cuenta el tiempo que a un paquete le toma desplazarse a lo largo del trayecto.
- Confiabilidad. Evalúa la probabilidad de falla en el enlace, calculada a partir del conteo de errores en la interfaz o de fallas previas en dicho enlace.
- Costo. Un valor determinado por el Sistema Operativo o por el administrador de la red que indica la preferencia por una ruta. El costo puede representar una métrica, una combinación de métricas o una política.

Balanceo de carga.

¿Qué pasa cuando dos o más rutas hacia el mismo destino tienen valores de métrica idénticos? ¿Cómo va a decidir el *router* qué ruta elegir para reenviar el paquete? En este caso, el *router* no selecciona sólo un trayecto. En vez de eso, el *router* balancea la carga entre las rutas con igual métrica.

Distancia administrativa.

Antes de determinar qué ruta utilizar cuando reenvían un paquete, los *routers* deben primero decidir qué rutas incluir en su tabla de *routing*. Pueden darse casos en los que aprendan una ruta hacia una red remota desde más de una fuente. Por ejemplo, una ruta estática pudo haber sido configurada hacia el mismo segmento de red que fue aprendido por un protocolo de *routing* dinámico.

Aunque es menos común, una red también puede comunicarse gracias al uso de más de un protocolo de *routing*. De ser así, en algunas situaciones será inevitable que el *router* aprenda rutas hacia las mismas direcciones de red por medio de diferentes protocolos. El hecho de que los protocolos no toman en cuenta las mismas propiedades para establecer sus métricas no permite comparar entre ellas para seleccionar la mejor. El proceso de *routing* necesitará determinar de alguna manera qué fuente usar y la distancia administrativa es parámetro indicado para resolver este problema.

Cada fuente de *routing* −protocolos, rutas estáticas e incluso redes directamente conectadas a una interfaz− es priorizada usando un valor de distancia administrativa. Ésta es un valor entero de 1 byte, es decir, que su valor decimal varía entre 0 y 255; mientras más bajo sea el valor, más preferencia tiene la fuente de *routing*. Una distancia de 0 es la mejor posible y de forma natural está asignado únicamente a las redes directamente conectadas a una interfaz del dispositivo. Una distancia administrativa con valor 255 indica que el *router* no creerá en la fuente de la ruta y no será incluida en la tabla de *routing*. La Tabla 2.3 contiene las fuentes de *routing* más comunes y sus respectivos valores de distancia administrativa.

Fuente de *routing*	**Distancia administrativa**
Conectada	0
Estática	1
EIGRP (resumida)	5
BGP externo	20
EIGRP Interno	90
IGRP	100
OSPF	110
IS-IS	115
RIP	120
EIGRP Externo	170
BGP Interno	200

Tabla 2.3. Valores de distancia administrativa para las principales fuentes de routing.

2.5.2 BGP (*Border Gateway Protocol*).

BGP es el protocolo de *routing* EGP utilizado en Internet. La versión 1 (RFC 1105) salió a la luz en 1989 para sustituir a EGP, que hasta entonces era el único protocolo de *routing* inter-dominio disponible. Cabe mencionar que EGP es un protocolo que cae dentro de la clasificación que lleva el mismo nombre. Posteriormente evolucionó a la versión 2 en 1990 (RFC 1163) y luego a la versión 3 en 1991 (RFC 1267). Finalmente, apareció la versión 4 (RFC 1771 y RFC 4271) que proporciona soporte para CIDR (*Classless Interdomain Routing*).

BGP funciona estableciendo *sockets* TCP hacia el puerto 179 y permite el intercambio dinámico de rutas conocidas entre los distintos AS a los que pertenecen los *routers* de la sesión. Este funcionamiento facilita una comunicación fiable, gracias a las bondades de un protocolo orientado a conexión como TCP.

Tomando en cuenta que en cada AS se utiliza algún IGP que, por supuesto, puede tener una definición distinta para obtener la métrica de las rutas aprendidas, es imposible encontrar el camino más conveniente hacia cada destino. Por ello, BGP utiliza un algoritmo llamado *path-vector* para seleccionar aquellas rutas que impliquen atravesar el menor número de Sistemas Autónomos posible.

En las tablas de *routing* determinadas por BGP, las rutas especifican una secuencia de números que identifican a los Sistemas Autónomos, los cuales deben seguirse en el orden indicado para que el paquete llegue a su destino. El último número de AS en la ruta corresponde al de la organización que tiene bajo su administración la red buscada; en otras palabras, es el AS donde se encuentra el destino. La razón principal que tiene el algoritmo de BGP para almacenar la trayectoria completa de los AS es la detección y eliminación de *loops*. Con ello se evita que, al reenviar los paquetes, éstos pasen varias veces por un mismo AS sin poder alcanzar su objetivo.

En cada sesión de BGP participan sólo dos *routers*, también llamados vecinos o *peers*. Dentro del dominio de una red pueden existir muchas sesiones BGP concurrentes, al igual que un solo enlace puede participar en varias sesiones BGP. En una sesión BGP es precisamente donde un *router* informa a otro sobre las redes que conoce y que sus vecinos, a su vez, podrán alcanzar a partir de él.

Cuando un *router* anuncia un prefijo a uno de sus vecinos, esa información es considerada válida hasta que dicho *router* anuncia explícitamente que ya no lo es o hasta que la sesión se pierde. Eso significa que en BGP no es necesario que la información de *routing* se actualice periódicamente. De tal forma que en un principio, cuando la sesión BGP se establece, existirá un gran flujo de mensajes; pero después de cierto tiempo de estabilización, los *routers* sólo necesitarán informar a sus vecinos sobre los cambios que han ocurrido.

Para almacenar la información de *routing*, BGP necesita un conjunto de tablas de datos denominadas RIB (*Routing Information Base*). Existen tres tablas diferentes, cuya función se explica enseguida:

- Adj-RIB-in: En ella se almacenan los prefijos aprendidos de un *peer* en particular. Hay tantas tablas de este tipo como vecinos de BGP se tengan.
- Loc-RIB: Almacena las mejores rutas seleccionadas que conoce el proceso BGP, ya sea porque las ha obtenido de la tabla de *routing* o bien porque se han aprendido a través de sesiones BGP. Hay sólo una por cada Sistema Autónomo.
- Adj-RIB-out: Almacena prefijos que pueden ser anunciados a otros vecinos. Esta tabla se construye a partir de la información contenida en la tabla Loc-RIB. Se tiene una tabla de este tipo por cada vecino BGP.

El diagrama de la Figura 2.10 ilustra el lugar que ocupan las RIB en el proceso de BGP.

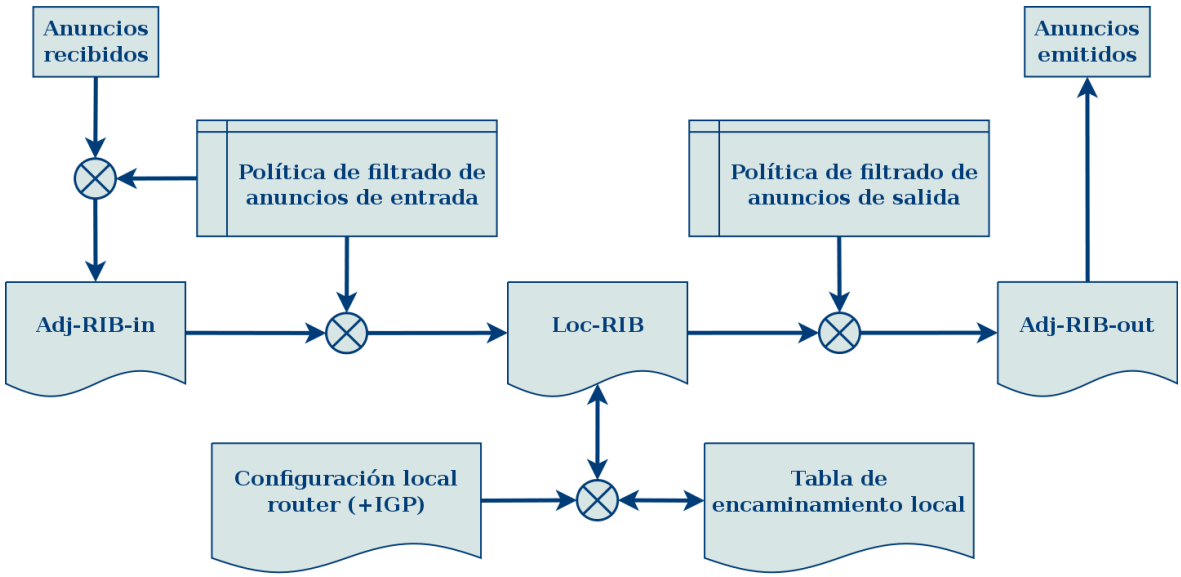

Figura 2.10. Procesamiento de los anuncios de BGP en un router.

Los mensajes intercambiados entre los integrantes de una sesión BGP pueden clasificarse en 4 tipos:

- OPEN. Éste es el primer mensaje que se envía después de que la conexión TCP se ha establecido. En él se incluye información interesante para los vecinos, tal como la versión del protocolo y el identificador del AS. Se incluye también, entre otros datos que podrían llegar a ser útiles, el tiempo durante el cual la sesión se mantendrá activa, el intervalo de envío de los mensajes de *keepalive* y el *hold time*.

- KEEPALIVE. Confirma que la sesión de BGP ha sido establecida y sirve para mantenerla activa. Así, en caso de que no haya una modificación en las tablas de *routing* de los dispositivos vecinos, sólo se intercambia éste tipo de mensaje de manera periódica. El intervalo por defecto es de 60 segundos entre cada uno, esperando un máximo de 3 mensajes *keepalive* sin respuesta antes de dar por terminada la sesión; de este modo, típicamente pasan 180 segundos antes de que los *routers* cierren el *socket* TCP y borren de su tabla de *routing* las rutas aprendidas desde el *router* vecino. A este tiempo máximo de espera se le llama *hold time*.
- NOTIFICATION. Se usa para cerrar una sesión BGP y la conexión TCP que la soporta. Con el mensaje se envía un código que indica el motivo del cierre de sesión. La consecuencia del cierre de la sesión BGP es la anulación de todas las rutas aprendidas en ella.
- UPDATE. Sirve para intercambiar información de *routing*, como el descubrimiento de nuevos destinos de red, el conjunto de atributos que acompaña a cada ruta, la eliminación de aquellas que ya no están activas, etcétera. La generación y recepción de estos mensajes ocurren solamente cuando se establece una vecindad entre dos *routers* o cuando se presenta un cambio en la tabla de *routing* de alguno de ellos, que a su vez implica modificaciones en las RIB de todos los dispositivos involucrados.

El proceso de BGP se compone por un sistema de 6 estados con un total de 13 eventos posibles. El intercambio de mensajes es la base para la interacción con otros equipos que ejecutan el mismo proceso. Los estados que componen el proceso de BGP están presentes en la Figura 2.11.

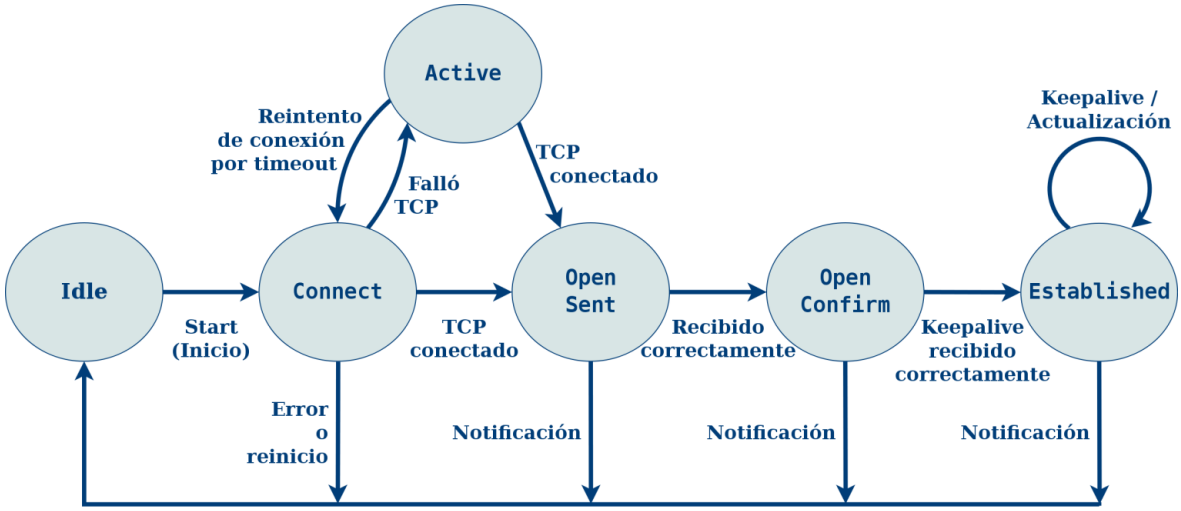

Figura 2.11. Estados que componen el proceso de BGP.

El significado de los estados disponibles se explica a continuación:

- Idle. En este estado, BGP rechaza todas las conexiones entrantes y ningún recurso local es asignado al *peer*. En respuesta al evento Start (comenzado ya sea por el sistema o por el administrador) el sistema local inicializa todos los recursos de BGP, se pone en marcha el temporizador para reintentos de conexión, se inicia una conexión de transporte con el vecino BGP y también espera por una conexión que puede ser iniciada por un *peer* remoto. Posteriormente el estado cambia a Connect. El valor exacto del temporizador para reintentos de conexión es un asunto local, pero debe ser lo suficientemente grande para permitir la inicialización de TCP.
- Connect. Aquí BGP está esperando por la conexión del protocolo de transporte para ser completada. Si la conexión del protocolo de transporte es exitosa, el sistema local limpia el temporizador para reintentos de conexión, completa la inicialización, envía un mensaje OPEN a su *peer* y cambia su estado a OpenSent. Si, por el contrario, la conexión del protocolo de transporte falla, el sistema local reinicia el temporizador para reintentos de conexión, continua esperando por una conexión que pueda ser iniciada por el *peer* de BGP remoto y cambia su estado a Active.
- Active. En este estado BGP está intentando contactar a un *peer* de BGP para iniciar una conexión con el protocolo de transporte. Si dicha conexión se establece, el sistema local limpia el temporizador para reintentos de conexión, completa la inicialización, envía un mensaje OPEN a su *peer*, se ajusta el "*Hold Time*" a un valor grande (el RFC 1771 sugiere 4 minutos) y cambia su estado a OpenSent. El evento Start es ignorado en el estado Active; en respuesta a cualquier otro evento (iniciado por el sistema o por el administrador) el sistema local libera todos los recursos de BGP asociados con esta conexión y cambia su estado a Idle.
- OpenSent. En este estado BGP espera por un mensaje OPEN de su *peer*. Cuando un mensaje OPEN es recibido, se verifica que todos los campos coincidan con exactitud. Si se detecta un error en la cabecera del mensaje, en el mensaje mismo o existe una colisión de conexiones, el sistema local envía un mensaje de tipo NOTIFICATION y cambia su estado a Idle. De no existir errores en el mensaje OPEN, BGP envía un mensaje KEEPALIVE y establece el período entre envíos de *keepalive*. El *Hold Time* que ya había sido establecido se reemplaza con un nuevo valor negociado por los *peers* y se revisan algunos otros parámetros como el identificador de AS. Al final del proceso, el estado es cambiado a OpenConfirm.

- OpenConfirm. En este estado BGP espera por un mensaje de tipo KEEPALIVE o NOTIFICATION. Si el sistema local recibe el *keepalive*, cambia su estado a Established. Si el *Hold Timer* termina antes de que el mensaje de KEEPALIVE sea recibido, el sistema local envía un mensaje de tipo NOTIFICATION con el código de error correspondiente y el estado cambia a Idle. Si sólo el temporizador de *keepalive* es el que se agota, el sistema local manda uno propio y reinicia el temporizador. Por otro lado, si una notificación de desconexión es recibida, el sistema local cambia su estado a Idle. En respuesta a un evento Stop (iniciado por el sistema o por el administrador), el sistema local envía un mensaje tipo NOTIFICATION con el código de error que lo generó y también cambia su estado a Idle.
- Established. Aquí, BGP puede intercambiar mensajes tipo UPDATE, NOTIFICATION y KEEPALIVE con su *peer*. Si el sistema local recibe una actualización o un *keepalive*, reinicia su *Hold Timer*. Contrario a esto, si recibe una notificación, el estado cambia a Idle. Si el temporizador de *keepalive* termina, el sistema local manda uno propio y reinicia el temporizador; lo mismo pasa cada que recibe o envía una actualización. En respuesta a un evento Stop (iniciado por el sistema o por el administrador), el sistema local envía un mensaje tipo NOTIFICATION con el código de error correspondiente y cambia su estado a Idle. Siempre que BGP pasa de Established a Idle, finaliza la conexión BGP –y también la de transporte− liberando todos los recursos asociados a ella y borrando todas las rutas derivadas de dicha vecindad.

Además de las sesiones que se establecen entre dispositivos de diferentes AS, los *routers* que pertenecen a un mismo dominio de *routing* deben intercambiar información para que su base de datos sea consistente y esté sincronizada. Para este propósito, existen dos protocolos que integran BGP y al mismo tiempo se complementan: iBGP (*internal* BGP) y eBGP (*external* BGP). Ambos fueron definidos en la versión 4. El mapeo de estos dos protocolos se ejemplifica en la Figura 2.12.

Cada *router* frontera establece una sesión iBGP con los demás, dentro de un mismo Sistema Autónomo. Aunque físicamente los *routers* no estén conectados de forma directa, la sesión iBGP se puede establecer indirectamente por ser un protocolo con funciones en capa 3 y 4 del modelo OSI, permitiendo que los mensajes puedan ser encaminados por el resto de *routers* operando en la red.

Para que todos los *routers* pertenecientes a un mismo AS conozcan las redes propagadas por iBGP, es necesario que la configuración esté diseñada para una topología de malla completa (*fully meshed*), es decir, que exista una sesión iBGP entre todos los *routers* operando con BGP. De este modo, una red con "n" *routers* necesitará de $\frac{n(n-1)}{2}$ sesiones iBGP.

En un Sistema Autónomo con un número elevado de *routers*, podría ser inconveniente tener tantas conexiones TCP abiertas permanentemente entre todos los nodos. Afortunadamente, en estos casos iBGP puede ser escalable para reducir el número de sesiones utilizando dos posibles configuraciones: BGP *Confederations* y *Route Reflectors*. Para cumplir los objetivos de este proyecto, no fue necesario profundizar hasta ese punto.

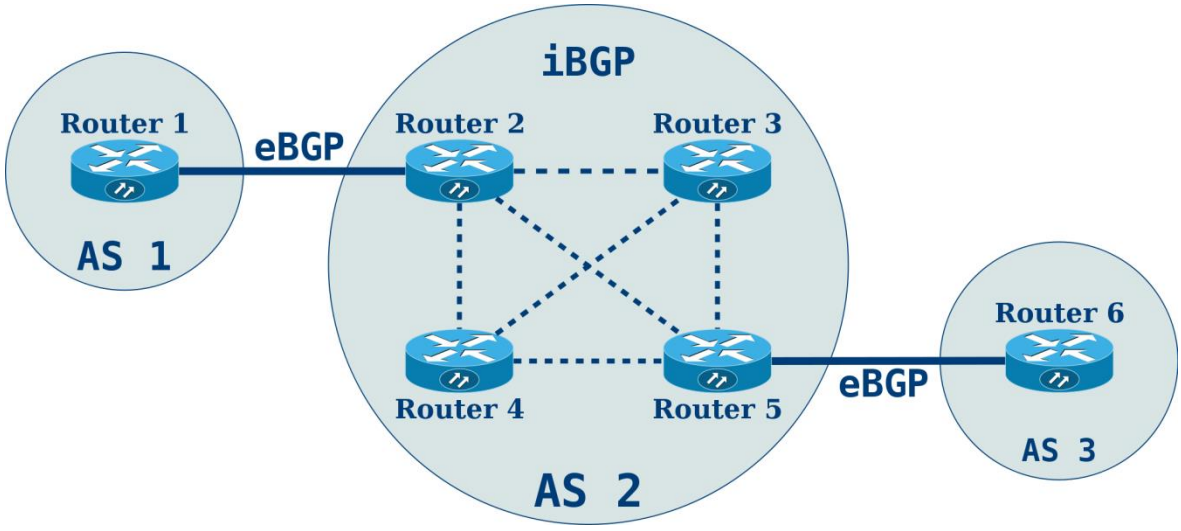

Figura 2.12. Casos de uso de iBGP y eBGP.

En el caso de una sesión eBGP se tiene establecida una conexión TCP entre dos *routers* frontera de dos Sistemas Autónomos distintos. Si el enlace que les da comunicación se viene abajo, ambos se darán cuenta en el instante que no puedan intercambiar los mensajes propios del protocolo. En tal caso, la sesión eBGP se considera finalizada una vez que el *Hold Time* se agota.

Es deseable seguir las siguientes recomendaciones para las sesiones eBGP, sobre todo si se trata del establecimiento de comunicación entre un AS cliente con un ISP:

- Generación de rutas agregadas estables. Se debe considerar resumir todos los prefijos internos del AS en pocos segmentos que los engloben. Esta estrategia permite reducir el número de rutas anunciadas y con ello se obtiene una mayor estabilidad con un menor uso de recursos computacionales.
- Configuración de políticas de entrada. Es sumamente recomendable aplicar reglas que filtren las actualizaciones y las rutas que se reciben provenientes de vecinos externos. De otro modo, se es propenso a recibir rutas hacia redes que no deseamos alcanzar a través de ciertos enlaces. Utilizando políticas de entrada también es posible balancear la carga de entre dos o más enlaces cuando se tienen múltiples conexiones entre los mismos Sistemas Autónomos.

- Configuración de políticas de salida. Filtrar los prefijos que serán anunciados hacia otros dominios de *routing* ayuda a protegerse contra errores de configuración. Con ello se evita anunciar rutas hacia redes no deseadas a través de la propia e incluso se puede forzar a los *routers* vecinos a que aumenten o disminuyan su preferencia de envío por ciertos enlaces.
- Configuración de *multihoming*. El término *multihoming* se refiere a la conexión de un cliente con dos o más ISP para crear o aumentar la redundancia y con ello disponer de distintos caminos hacia un destino de Internet, de tal manera que pueda seleccionarse el más conveniente de acuerdo a las necesidades que se tengan. Esta característica no fue necesaria para el desarrollo del proyecto.

Atributos de BGP.

A partir del mensaje UPDATE es posible distinguir una buena cantidad de atributos que acompañan al prefijo que se comparte como ruta. Dichos atributos son utilizados para insertar las rutas en las RIBs y para modificar la preferencia de la trayectoria que seguirán los paquetes hacia un destino. De igual forma, con la manipulación de estos parámetros se pueden aplicar reglas de filtrado a los mensajes BGP que serán anunciados y recibidos dentro de una sesión. Los atributos obligatorios son los siguientes:

- ORIGIN. Representa la fuente desde la que fue aprendida la ruta: indica una "i" si la ruta proviene de un IGP, una "e" si se aprendió por EGP o un signo "?" (INCOMPLETE) en caso de que el origen sea desconocido. Este último indicador normalmente es consecuencia de la redistribución de rutas estáticas.
- AS-PATH. Cada Sistema Autónomo agrega su número identificador mediante este atributo en cada una de las rutas que aprende antes de reenviarlas hacia otro dominio de *routing*. Como resultado, cada ruta tendrá una lista con los números de los AS por los que un mensaje tendrá que pasar antes de llegar a su destino. Otra utilidad del AS-PATH es eliminar la posibilidad de *loops* y su valor se usa como referencia para el filtrado de rutas según las políticas de *routing*. Un *router* deberá ignorar la recepción de un anuncio de ruta si este atributo ya contiene el número de su propio AS.
- NEXT-HOP. Este atributo indica la dirección IP del siguiente nodo al que habrá que reenviar el paquete para que llegue a su destino. Su información es útil en caso de que el siguiente nodo no utilice BGP y permite concebir la topología compuesta por BGP de una forma independiente a la topología física de la red, ya que la dirección IP del siguiente salto no necesariamente tiene que ser la del próximo nodo por el que pasará físicamente la información.

Además de los ya mencionados, existe un conjunto de atributos reconocidos por el protocolo pero que no es obligatorio incluirlos en la información que contienen los mensajes. Algunos de ellos se explican en la siguiente lista:

- LOCAL_PREFERENCE. Sólo es anunciado en las sesiones iBGP. Es un parámetro utilizado localmente dentro de un Sistema Autónomo y su utilidad es priorizar ciertas rutas que se anuncian internamente.
- ATOMIC_AGGREGATE. Especifica que la ruta anunciada es realmente una ruta agregada por una fuente externa al proceso natural de BGP y que pudiera no ser originada por el Sistema Autónomo que la está anunciando. Las rutas agregadas utilizando este atributo técnicamente no existen, pero representan o resumen a un conjunto de direcciones en una sola.
- MED (*Multi-Exit Discriminator*). Es útil cuando se tienen dos Sistemas Autónomos interconectados por más de un *router* frontera en cada uno o por más de un enlace. Su función es muy parecida a la métrica utilizada en los IGP. A grandes rasgos, el valor que se configure al atributo servirá para dar preferencia a las rutas aprendidas por un enlace respecto a las que se aprendieron mediante otro en el *router* vecino. El valor del atributo MED es compartido a través de sesiones eBGP y no se conserva cuando la ruta se redistribuye a un AS diferente.
- WEIGHT. Se utiliza como el criterio de selección más importante cuando se tienen varias rutas hacia una misma red. Es un parámetro que sólo es válido localmente en cada *router*, de modo que no se propaga en los anuncios hechos a cualquiera de los vecinos. Se representa con un número de dos bytes, así que puede variar entre 0 y 65535 decimal; mientras más alto sea su valor, mayor preferencia tiene la ruta. Por defecto es igual a 32768 en las rutas originadas en el propio *router* y 0 para las demás rutas.

Proceso de decisión.

La metodología de BGP tiene la tarea de recibir, procesar, seleccionar y distribuir los mensajes necesarios para funcionar de acuerdo a la configuración que hizo el administrador de la red en cada *router*. El procesamiento de la información de *routing* que mantiene la consistencia en las RIB tiene fundamento en una serie de criterios técnicos y administrativos.

Adicional a lo anterior, debe existir sincronía entre los *routers* que pertenecen a un mismo AS. Esto es, las tablas de *routing* de todos ellos deben ser consistentes. De no ser así, no se anunciará una ruta por eBGP antes de que todos los *routers* del AS la hayan aprendido por IGP.

También es importante considerar que la dirección IP presente en el atributo NEXT-HOP sea alcanzable. Si esa dirección no se incluye expresamente en la tabla de *routing* o

no cae en algún segmento de los ahí presentes, el proceso natural de BGP no será capaz de considerarla como válida.

Al momento de elegir entre dos o más rutas disponibles localmente en un *router* para alcanzar un mismo segmento de red, el protocolo revisa algunos criterios de acuerdo a un orden de importancia. A este procedimiento se le conoce como "Algoritmo de selección de mejor trayectoria de BGP". El proceso asigna la primera ruta válida como la mejor ruta actual, luego compara dicha ruta con la siguiente en la lista hasta que termina con todas las rutas válidas. La siguiente lista contiene las reglas que son utilizadas para elegir la mejor ruta –en un ambiente sin *Route Reflectors*[10]– en orden de mayor a menor importancia:

1) La ruta con el valor mayor de WEIGHT es seleccionada.
2) Se prefiere a la trayectoria con el mayor valor de LOCAL_PREFERENCE.
3) Tiene prioridad la ruta que fue localmente originada por medio de un subcomando BGP como "network" o "aggregate" o a través de redistribución por un IGP.
4) Se toma en cuenta el camino con menor AS_PATH.
5) BGP prefiere la ruta con el tipo de origen menor. Un IGP es menor que un EGP; y un EGP es menor que INCOMPLETE.
6) Selecciona la ruta con el MED más bajo.
7) Las rutas aprendidas por eBGP tienen mayor preferencia que las de iBGP, por su distancia administrativa.
8) Menor métrica de IGP hacia el siguiente salto. Se selecciona la ruta con el NEXT-HOP más próximo, de acuerdo a la métrica del IGP utilizado en la red dentro del mismo AS.
9) Determina si se requiere la instalación de múltiples rutas en la tabla de *routing* para balancear carga (BGP Multipath).
10) Cuando ambas trayectorias son externas –conocidas por eBGP–, BGP prefiere a la que fue recibida primero (la más antigua).
11) Se toma en cuenta a la ruta que proviene del *router* BGP con el ID más pequeño.
12) BGP selecciona la ruta proveniente del vecino con la dirección IP más baja.

[10] Los *Route Reflectors* de BGP son clusters de routers implementados cuando las redes de los sistemas autónomos son muy extensas, con el objetivo de reducir la cantidad de tráfico debido al protocolo mismo, las sesiones TCP abiertas en cada router por cada vecino existente y, en consecuencia, reducir el procesamiento y uso de recursos en los routers.

2.6 VRRP (*Virtual Router Redundancy Protocol*).

Existe cierto número de métodos que un *host* final (usuario final) puede utilizar para determinar su *router* de primer salto hacia ciertas direcciones IP en particular. Estos incluyen ejecutar un protocolo de *routing* dinámico como RIP u OSPF o usar una ruta estática por defecto.

Ejecutar un protocolo de *routing* dinámico en cada *host* final puede no ser factible por un buen número de razones, incluyendo el tiempo y costos de administración que tienen que invertirse en cada *host*, el consumo de recursos y procesamiento, riesgos de seguridad y la falta de herramientas para implementar estos protocolos en algunas plataformas. El descubrimiento de *routers* vecinos puede requerir la participación activa de todos los *hosts* de la red, lo que implica grandes cantidades de tiempo transcurrido antes de que la red converja. Eso puede significar un considerable retraso en la detección de un nodo en falla y conducir a periodos inaceptables con la red en un estado de "agujero negro".

El uso de una ruta estática por defecto es bastante popular y minimiza la configuración y los costos de procesamiento en el *host* final, además de ser virtualmente soportada por todas las implementaciones IP. También está la posibilidad de usar DHCP, un protocolo diseñado para configurar automáticamente a los *hosts* finales, dotándolos de una dirección IP válida y una ruta hacia un *router* por defecto. Sin embargo, ambas soluciones propician la existencia de un punto único de falla: la pérdida del *default gateway* significa un evento catastrófico, aislando a todos los nodos que no son capaces de detectar ningún camino alternativo que esté disponible.

VRRP está diseñado para eliminar el punto único de falla inherente en el ambiente antes descrito. VRRP especifica un protocolo de elección que asigna dinámicamente la responsabilidad de ser un *router* virtual a uno de los *routers* VRRP de la LAN. El *router* VRRP que administra las direcciones IP asociadas con un *router* virtual es llamado "maestro" y se encarga de recibir y reenviar los paquetes que se encaminan utilizando esa dirección IP virtual como primer salto. El proceso de elección proporciona dinámicamente una conmutación por falla en la responsabilidad del reenvío de paquetes una vez que el maestro sale de línea. Cualquiera de las direcciones IP virtuales de los *routers* puede ser utilizada como el *default gateway* de los usuarios. La ventaja al utilizar VRRP es que se adquiere una disponibilidad más alta del *default gateway* sin que eso precise la configuración de un protocolo de *routing* dinámico en cada *host*.

VRRP realiza una función muy similar a los protocolos propietarios HSRP (*Hot Standby Router Protocol*) e IPSTB (IP *Standby Protocol*).

Todo el mensajeo de VRRP utiliza datagramas IP hacia direcciones de *multicast*, así que el protocolo funciona sobre la variedad de tecnologías LAN multi-acceso que soporten IP *multicast*. Cada *router* virtual tiene una única y bien conocida dirección MAC asociada. La dirección MAC del *router* virtual es utilizada como la fuente en todos los mensajes VRRP que se envían periódicamente por el *router* maestro.

Un *router* virtual está definido por su identificador de *router* virtual (VRID: *Virtual Router IDentifier*) y un conjunto de direcciones IP virtuales. Un *router* VRRP puede asociar un *router* virtual con su dirección real en una interfaz y puede ser también relacionado con configuraciones de *routers* virtuales adicionales con su respectiva prioridad. La relación entre el VRID y las direcciones debe la misma en todos los *routers* VRRP de la LAN. Sin embargo, no existe restricción contra el reúso de un VRID con un mapeo de direcciones diferentes en LANs diferentes. El alcance de cada *router* virtual está restringido a una sola LAN.

Para minimizar la cantidad de tráfico en la red, sólo el maestro de cada *router* virtual envía periódicamente mensajes de tipo "VRRP *Advertisement*". Un *router* de respaldo no intentará tomar el lugar del maestro a menos que tenga una prioridad más alta.

2.7 EoIP (*Ethernet over Internet Protocol*).

Ethernet over IP *Tunneling* es un protocolo de MikroTik RouterOS que crea un túnel Ethernet entre dos *routers* sobre una conexión IP. El túnel EoIP puede establecerse sobre túneles IP-IP, túneles PPTP o cualquier otra conexión capaz de transportar información IP.

Cuando en dos *routers*, comunicados entre sí por un enlace WAN, la función de *bridge* (modo puente) está activa, todo el tráfico será tratado y encapsulado justo como si existiera una interfaz física de Ethernet y un cable conectado entre ellos.

Entre las ventajas que se pueden tener en los *routers* con interfaces EoIP habilitadas, se encuentran:

- Posibilidad de encapsular tráfico de redes de área local en Internet.
- Posibilidad de encapsular tráfico de redes de área local en túneles cifrados.
- Posibilidad de encapsular tráfico de redes de área local en redes inalámbricas ad-hoc 802.11b.

El protocolo EoIP encapsula tramas de Ethernet en paquetes GRE, tal como lo haría en PPTP, y los envía al extremo remoto del túnel.

Notas importantes de EoIP:

El parámetro Tunnel-ID es el método utilizado para identificar un túnel. Éste identificador debe ser único para cada túnel EoIP.

La MTU (*Maximum Transmission Unit*) debe ser fijada a 1500 bytes para evitar la fragmentación de paquetes dentro del túnel, el cual permite el encapsulamiento transparente en capa 2 tal como en redes Ethernet, así que debe ser posible transportar tramas Ethernet completas a través del túnel.

Es altamente recomendable establecer una única dirección MAC en cada túnel para que los algoritmos de encapsulamiento en capa 2 trabajen correctamente. Para interfaces EoIP se pueden emplear direcciones MAC que estén en el rango 00:00:5E:80:00:00 - 00:00:5E:FF:FF:FF, que la IANA tiene reservadas para tales casos.

Capítulo 3.

Software y Middleware.

Durante los capítulos pasados se cubrió lo necesario para entender el diseño y la operación tanto de la redundancia WAN como del *cluster* construidos para el proyecto. Ahora es el turno de explicar las características de las herramientas que hicieron posible la puesta en marcha de la solución y cómo se fundamentan en todos los principios teóricos ya mencionados.

3.1 Debian.

El proyecto Debian está conformado por una asociación de personas que han hecho causa común para crear un Sistema Operativo libre y que fue bautizado con el mismo nombre del proyecto.

Un Sistema Operativo es un conjunto de programas y utilidades básicas que hacen que una computadora funcione. El centro de un Sistema Operativo es el núcleo, también llamado kernel. El kernel es el programa más importante en la computadora, realiza todo el trabajo básico y le permite ejecutar otros programas.

Los sistemas Debian actualmente usan el núcleo de Linux o de FreeBSD. Linux es una pieza de *software* creada en un principio por Linus Torvalds y desarrollada por miles de programadores a lo largo del mundo. FreeBSD es un Sistema Operativo que incluye un núcleo y otro *software*.

Sin embargo, se está trabajando para ofrecer Debian con otros núcleos, en especial con Hurd. El Hurd es una colección de servidores que se ejecutan sobre un micro-núcleo (como Mach) para implementar distintas funcionalidades. Hurd es *software* libre producido por el proyecto GNU.

Una gran parte de las herramientas básicas que completan el Sistema Operativo vienen del proyecto GNU; de ahí los nombres: GNU/Linux, GNU/kFreeBSD, y GNU/Hurd. Estas herramientas también son libres.

Desde luego, lo que la gente quiere es el *software* de aplicación: herramientas que los ayuden a realizar lo que necesiten hacer, desde editar documentos y ejecutar aplicaciones de negocios, hasta divertirse con juegos y escribir más *software*. Debian viene con más de 37500 paquetes (*software* pre-compilado y empaquetado en un formato amigable para una instalación sencilla), un gestor de paquetes (APT) y otras utilidades que hacen posible gestionar miles de paquetes en miles de computadoras de manera tan fácil como instalar una sola aplicación. Todos ellos de forma gratuita.

Debian estaba pensada para ser desarrollada cuidadosa y conscientemente; y ser mantenida y soportada con un cuidado similar. Comenzó con un pequeño grupo de hackers de *software* libre y fue creciendo gradualmente hasta convertirse en una gran comunidad de desarrolladores y usuarios bien organizada. Es la única distribución que está abierta a las contribuciones de cada desarrollador y usuario que deseen participar con su trabajo. Es la única distribución relevante de Linux que no es una entidad comercial y es el único gran proyecto con una constitución, contrato social y documento de directrices que lo organizan. Debian es también la única distribución que se "micro-empaqueta" y que utiliza una detallada información de las dependencias de cada paquete con respecto a otros para asegurar la consistencia del sistema cuando tiene lugar una actualización.

Debian ha adoptado un gran conjunto de directrices y procedimientos para el empaquetamiento y la distribución de *software* para poder alcanzar y mantener altos estándares de calidad. Se producen herramientas, sistemas automáticos y documentación de cada uno de los aspectos claves de Debian de una forma abierta y visible para poder sostener estos estándares. Lo producen cerca de un millar de desarrolladores activos, dispersos por el mundo que ayudan voluntariamente en su tiempo libre. Son pocos los desarrolladores que realmente se han encontrado en persona. La comunicación se realiza principalmente a través de correo electrónico (listas de correo en lists.debian.org) y a través de IRC (canal #debian en irc.debian.org).

Aunque no hay disponibles estadísticas precisas (ya que Debian no requiere que los usuarios se registren), hay signos bastante evidentes de que Debian es usado por un amplio número de organizaciones, grandes y pequeñas, así como muchos miles de personas de forma individual. Instituciones educativas como la UNAM, University of Cambridge, Universidad Politécnica de Madrid, Czech Technical University, St. Petersburg State Polytechnic University, Harvard University, Massachusetts Institute of Technology, empresas de calidad reconocida mundialmente, así como diversas organizaciones gubernamentales pertenecientes a Estados Unidos, Francia, Rusia, Alemania, Italia, Brasil, entre otros, han declarado públicamente el uso de sistemas operativos Debian para el desarrollo de algunas de sus actividades más importantes. Información más detallada se puede encontrar en el URL: https://www.debian.org/users/

3.2 Cisco IOS.

Es el *software* utilizado en la mayoría de *routers* y *swtiches* Cisco (aunque los primeros dispositivos de esta marca funcionaban con CatOS). IOS es un paquete de funciones de telecomunicaciones integradas en un Sistema Operativo multitarea, que incluye procedimientos para *routing*, *switching* e interconexión entre redes.

IOS cuenta con una interfaz "modo texto" para interactuar con el usuario y facilitar la operación y la configuración del sistema a través de comandos relativamente simples y

jerarquizados. El grupo de comandos disponible está determinado por el modo y el nivel de privilegio del usuario que lo utiliza. El modo de configuración global permite comandos para cambiar la configuración del sistema, mientras que el modo de configuración de interfaz sirve para cambiar la configuración específica en una de ellas. Todos los comandos tienen asignado un nivel de privilegio que va desde el 0 hasta el 15 y sólo pueden ser ejecutados por los usuarios con un nivel de privilegio igual. A través de la CLI (*Command Line Interface*), los comandos disponibles para cada nivel de privilegio pueden ser definidos.

Cisco Systems define brevemente al IOS como un *software* de infraestructura de redes que integra innovación y servicios de carácter crítico para redes empresariales.

3.3 OpenSWAN.

Openswan es una implementación de IPsec de código abierto disponible en varios sistemas Linux. Emplea el protocolo de establecimiento de llaves IKE tanto en su versión 1 como en la versión 2 y se ejecuta como un demonio a nivel de usuario.

El *software* de Openswan comenzó como un derivado del ahora extinto proyecto FreeS/WAN (*Free Secure Wide-Area Networking*) y continúa usando la GNU GPL (*General Public License*). A diferencia de su predecesor, Openswan no es exclusivo de los sistemas operativos GNU/Linux.

Las interfaces de Openswan en conjunto con el kernel de Linux utilizan los enlaces de red para transferir las llaves de cifrado. El cifrado y el descifrado de paquetes se llevan a cabo al nivel del kernel de Linux.

Openswan utiliza las librerías criptográficas de los NSS (*Network Security Servers*), que son requeridas de conformidad con el FIPS (*Federal Information Processing Standard*) para lo referente a la seguridad en el manejo de la información.

3.4 Quagga.

Quagga es una *suite* de *software* para *routing* que provee implementaciones de OSPFv2, OSPFv3, RIPv1, RIPv2, RIPng y BGP-4 para plataformas UNIX, particularmente FreeBSD, Linux, Solaris y NetBSD. Quagga es un derivado de GNU Zebra, que fue desarrollado por Kunihiro Ishiguro.

Aterrizando más la descripción, se puede definir a Quagga como un paquete de *software* que provee servicios de *routing* a través de comunicación TCP/IP, gracias a protocolos de *routing* como los anteriormente mencionados. Soporta características especiales de BGP (como BGP *Route Reflectors*) y protocolos dedicados para IPv6.

Un sistema operando con Quagga intercambia información de *routing* con otros dispositivos. Utiliza esta información para actualizar la tabla de *routing* del kernel de manera que se entregue los datos correctos en el lugar correcto. Es posible cambiar dinámicamente la configuración y el contenido de la tabla de *routing* desde la terminal de Quagga.

El *software* tradicional para hacer *routing* está hecho como un proceso que otorga todas las funcionalidades de los protocolos de *routing*. Es sencillo agregar un nuevo demonio que se encargue de un protocolo de *routing* en específico sin que afecte a ningún otro *software*.

Una arquitectura multiproceso como la de Quagga brinda extensibilidad, modularidad y mantenibilidad; al mismo tiempo que permite administrar un archivo de configuración y una interfaz de terminal para cada protocolo (cada protocolo está asociado a un demonio diferente). Cuando se tienen una diversidad de protocolos de *routing* combinados con rutas estáticas en el mismo dispositivo, puede llegar a ser muy molesto manipular todos los archivos necesarios, por esta razón Quagga ofrece la posibilidad de concentrar toda la administración en una sola interfaz que conecta con cada demonio por medio de un *socket* de Unix y que funciona como un proxy de entrada para el usuario.

3.5 Pacemaker.

Pacemaker es un administrador de recursos de *cluster*. Logra obtener la máxima disponibilidad para todos ellos por medio de la detección de fallas acontecidas en el nodo o a nivel de recursos, haciendo uso de mensajes y afiliaciones provistas por una infraestructura de *cluster* (Corosync o Heartbeat) para su inmediata recuperación.

Dentro de las principales características de Pacemaker se incluyen:

- Detección de fallas y recuperación a nivel de nodo o a nivel de servicio.
- Almacenamiento agnóstico, ningún requisito de almacenamiento compartido.
- Recursos agnósticos, todo lo que puede ser estructurado y programado, puede incluirse en un *cluster*.
- Soporta STONITH (*Shoot The Other Node In The Head*) para asegurar la integridad de la información.
- Capacidad de controlar *clusters* grandes y pequeños.
- Soporta prácticamente cualquier configuración de redundancia.
- Configuración replicada automáticamente y que puede ser actualizada prácticamente desde cualquier nodo.
- Habilidad para crear un orden distribuido de los recursos en los miembros del *cluster*, detallando en dónde puede colocarse un recurso y dónde no puede

ejecutarse, así como la convivencia y la incompatibilidad de recursos en un mismo nodo del *cluster*.

- Soporte por tipos de servicio avanzados.
- Clones. Útil para servicios que necesitan estar activos en múltiples nodos.
- Multi-estado. Ideal para servicios con múltiples modos (por ejemplo, maestro/esclavo, primario/secundario).
- Herramientas de administración de *cluster* unificadas y programables.

Arquitectura de Pacemaker.

En el nivel más alto, un *cluster* está hecho de tres piezas:

- Componentes que no están en *cluster*. Estas piezas incluyen los recursos mismos, *scripts* que los arrancan, los detienen y los diagnostican.
- Administración de los recursos. Pacemaker ofrece el control que procesa y reacciona ante los eventos que acontecen en un *cluster*. Dichos eventos incluyen nodos que se agregan o que abandonan el *cluster*. Los eventos de los recursos son causados por fallas, mantenimientos, actividades calendarizadas u otras acciones administrativas. Pacemaker calculará el estado ideal del *cluster* y trazará un camino para conseguirlo después de cualquiera de estos eventos. Esto puede incluir mover recursos, detener nodos o incluso forzarlos a salir de línea mediante interrupciones de alimentación remotas.
- Infraestructura de bajo nivel. Proyectos como Corosync, CMAN y Heartbeat ofrecen mensajería fiable, afiliación (cuáles nodos están funcionando y cuáles no) e información acerca del quorum del *cluster*.

Cuando se combina con Corosync, Pacemaker también soporta *clusters* con sistemas de archivos de código abierto.

Debido a la estandarización dentro de la comunidad de sistemas de archivos en *cluster*, se hace uso de un administrador distribuido común que utiliza Corosync para gestionar los mensajes entre los nodos y las capacidades de afiliación y Pacemaker para controlar los servicios.

Componentes internos de Pacemaker.

Pacemaker en sí mismo se compone de 5 elementos clave, mismos que se mapean en la Figura 3.1 y que se mencionan en la siguiente lista:

1) CIB (*Cluster Information Base*).
2) CRMd (*Cluster Resource Management daemon*).

3) LRMd (*Local Resource Management daemon*).

4) PEngine (también conocido como PE o *Policy Engine*).

5) STONITHd.

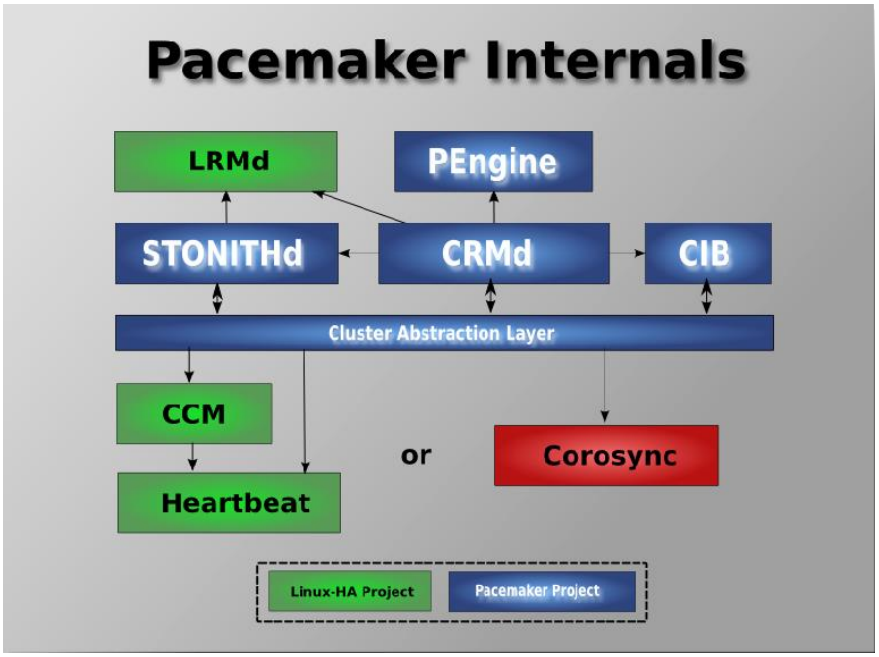

Figura 3.1. Elementos internos de Pacemaker.[11]

La CIB usa XML para representar tanto la configuración de *cluster* como el estado actual de todos los recursos en el *cluster*. Los contenidos de la CIB son sincronizados automáticamente a través de todo el *cluster* y son usados por el PEngine para calcular el estado ideal del *cluster* y cómo debería ser alcanzado.

La lista de instrucciones es entonces enviada al DC (*Designated Controller*). Pacemaker centraliza toda la toma de decisiones por medio de la elección de una de las instancias del CRMd para que actúe como el maestro. En caso de que el proceso CRMd elegido o el nodo entero fallen, uno nuevo es rápidamente establecido.

El DC lleva a cabo las instrucciones del PEngine en el orden requerido pasándolas ya sea por el LRMd propio o los CRMd de otros nodos a través de la infraestructura de mensajería del *cluster* (que a su vez las enviará a sus respectivos procesos de LRMd).

Todos los nodos del *cluster* reportarán los resultados de sus operaciones al DC y, basándose en los resultados esperados y los actuales, ejecutará cualquier acción que necesite para que la acción previa sea completada, o bien, abortará el proceso e instruirá al

[11] Documento: "Pacemaker 1.1 Configuration Explained. An A-Z guide to Pacemaker's Configuration Options". Edition 1. Copyright© 2009-2011 Andrew Beekhof.

PEngine para que recalcule el estado ideal del *cluster* basado en los resultados no esperados.

En algunos casos puede ser necesario apagar los nodos con el fin de proteger la información compartida o para completar la recuperación del recurso. Para esto, Pacemaker viene con STONITHd.

STONITH es un acrónimo para "*Shoot The Other Node In The Head*" y es usualmente implementado con un interruptor de encendido y apagado remoto. Es uno de los medios más comunes para hacer *fencing*.

En Pacemaker, los dispositivos de STONITH son modelados como recursos y configurados en la CIB para habilitarlos, logrando una fácil atención a las fallas. Sin embargo, STONITHd presta especial atención al entendimiento de la topología de STONITH, de tal manera que los clientes simplemente soliciten aislar o apagar un nodo y él hará el resto.

3.6 Corosync.

El motor de *clusters* Corosync es un grupo de sistemas de comunicaciones con características muy específicas para implementar alta disponibilidad dentro de las aplicaciones. El proyecto provee una Interfaz de Programación de Aplicaciones (API: *Application Programming Interface*) en C con cuatro características:

- Un modelo de comunicación grupal entre procesos cerrados con sincronía virtual que garantiza la creación de máquinas de estado replicadas.
- Un administrador de disponibilidad simple que restablece el proceso de la aplicación cuando éste falla.
- Una base de datos de configuración y de estadísticas en memoria que proporciona la habilidad de establecer, seleccionar y recibir notificaciones de cambios de información.
- Un sistema de quorum, que notifica a las aplicaciones cuando el quorum es alcanzado o se ha perdido.

El proyecto Corosync es usado como una herramienta para la alta disponibilidad por otros proyectos como Apache, Qpid y Pacemaker.

3.7 *Shell Script* en Linux.

Para abordar con mayor facilidad este tema, será necesario establecer primero el concepto de un *shell* en Linux.

En las primeras etapas de la computación, las instrucciones eran suministradas utilizando lenguaje binario, que es un sistema muy complicado de leer y escribir para casi todas las personas. En los sistemas operativos Linux existe un programa llamado "*shell*". El *shell* acepta instrucciones o comandos (la mayoría en idioma inglés) y, si son válidos, estos son pasados al kernel.

Shell es un programa de usuario o un ambiente desarrollado para la interacción con el usuario. Sirve como un intérprete de lenguaje de comandos que los ejecuta al leerlos desde la entrada estándar del sistema (regularmente el teclado) o desde un archivo.

El *shell* no es parte del kernel, pero sí utiliza el sistema del kernel para ejecutar programas, crear archivos, etcétera. Para usar un *shell* simplemente se tienen que escribir comandos, aunque realmente el usuario empieza a utilizarlo tan pronto como inicia su sesión en el sistema.

Una vez que se explicó qué es y para qué sirve un *shell*, se puede definir con más precisión qué es un "*shell script*". Normalmente los *shell* son interactivos, eso quiere decir que el *shell* acepta comandos de parte del usuario y los ejecuta, normalmente arrojando un resultado. Pero cuando se ejecuta una secuencia de 'n' número de comandos, entonces se puede almacenar dicha secuencia en un archivo de texto e instruir al *shell* para que lea y ejecute este archivo en lugar de que el usuario introduzca los comandos. Esto es conocido como *shell script*: una serie de comandos escritos en un archivo de texto plano.

¿Para qué elaborar un *shell script*? La anterior definición parece simple y hasta carente de sentido. Probablemente muchos usuarios no le encontrarían gran utilidad en una primera instancia. No obstante, su aplicación es verdaderamente valiosa por razones como las siguientes:

- Un *shell script* puede tomar instrucciones de entrada de parte del usuario o un archivo y reflejar el resultado en la pantalla.
- Muy útil para crear comandos propios.
- Ahorra gran cantidad de tiempo una vez que está completo.
- Automatiza muchas de las tareas que se realizan en el día a día.
- Parte de la administración del sistema puede ser automatizada.

Los *shell scripts* jugaron un papel fundamental en muchas herramientas y configuraciones utilizadas para este proyecto.

3.8 Apache2.

Apache HTTP Server Project es un esfuerzo por desarrollar y mantener un servidor HTTP de código abierto para sistemas operativos modernos, incluyendo Unix y Windows NT. El objetivo de dicho proyecto es proporcionar un servidor seguro, eficiente y extensible que ofrezca servicios HTTP en sincronía con los estándares HTTP actuales.

Apache ha sido el servidor web más popular en Internet desde abril de 1996 y de acuerdo a la documentación de su página oficial, tiene las siguientes características:

- El servidor HTTP Apache implementa los más recientes protocolos, incluyendo HTTP/1.1 (RFC2616).
- Es altamente configurable y extensible con módulos de terceros.
- Puede ser personalizado a través de otros módulos utilizando la API de Apache.
- Proporciona el código fuente completo y viene con una licencia sin restricciones.
- Se ejecuta en Windows 2000, Netware 5.x y superiores, OS/2 y la mayoría de las versiones de Unix, así como en varios otros sistemas operativos.
- Constantemente está siendo mejorado y bajo desarrollo.
- Propicia la retroalimentación del usuario mediante nuevas ideas, reportes de errores y parches.
- Implementa varias características de uso frecuente, incluyendo:
    - Bases de satos DBM, además de bases de datos relacionales y LDAP para autenticación.
    - Permite crear fácilmente páginas protegidas por contraseñas con un número enorme de usuarios autorizados, sin afectar el rendimiento del servidor.
    - Respuestas personalizadas a errores y problemas diversos.
    - Permite cargar archivos o incluso CGI *scripts*, que son regresados por el servidor en respuesta a errores o problemas detectados.
    - Múltiples directivas de índice de directorios.
    - Re-direccionamientos ilimitados de URL.
    - *Hosts* virtuales.
    - Formatos de registros configurables.

3.9 PostgreSQL.

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado y en sus últimas versiones no tiene nada que envidiarle a otras bases de datos comerciales.

PostgreSQL utiliza un modelo cliente/servidor y usa multiprocesos en vez de multi-hilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará al resto y el sistema continuará funcionando.

La Figura 3.2 tiene un esquema con los componentes más importantes en un sistema PostgreSQL, mismos que son descritos a continuación:

- Aplicación cliente: esta es la aplicación cliente que utiliza PostgreSQL como administrador de bases de datos. La conexión puede ocurrir vía TCP/IP o *sockets* locales.
- Demonio postmaster: este es el proceso principal de PostgreSQL. Es el encargado de escuchar por un puerto/*socket* las conexiones entrantes de clientes. También es el encargado de crear los procesos hijos.
- Archivos de configuración: los 3 archivos principales de configuración utilizados por PostgreSQL son: postgresql.conf, pg_hba.conf y pg_ident.conf
- Procesos hijos de postgres: procesos hijos que se encargan de autenticar a los clientes, de gestionar las consultas y mandar los resultados a las aplicaciones clientes.
- PostgreSQL *share buffer cache*: memoria compartida usada por PostgreSQL para almacenar datos en caché.
- *Write-Ahead Log* (WAL): componente del sistema encargado de asegurar la integridad de los datos –recuperación de tipo "redo" –.
- *Kernel disk buffer cache*: caché de disco del Sistema Operativo.
- Disco: disco físico donde se almacenan los datos y toda la información necesaria para que PostgreSQL funcione.

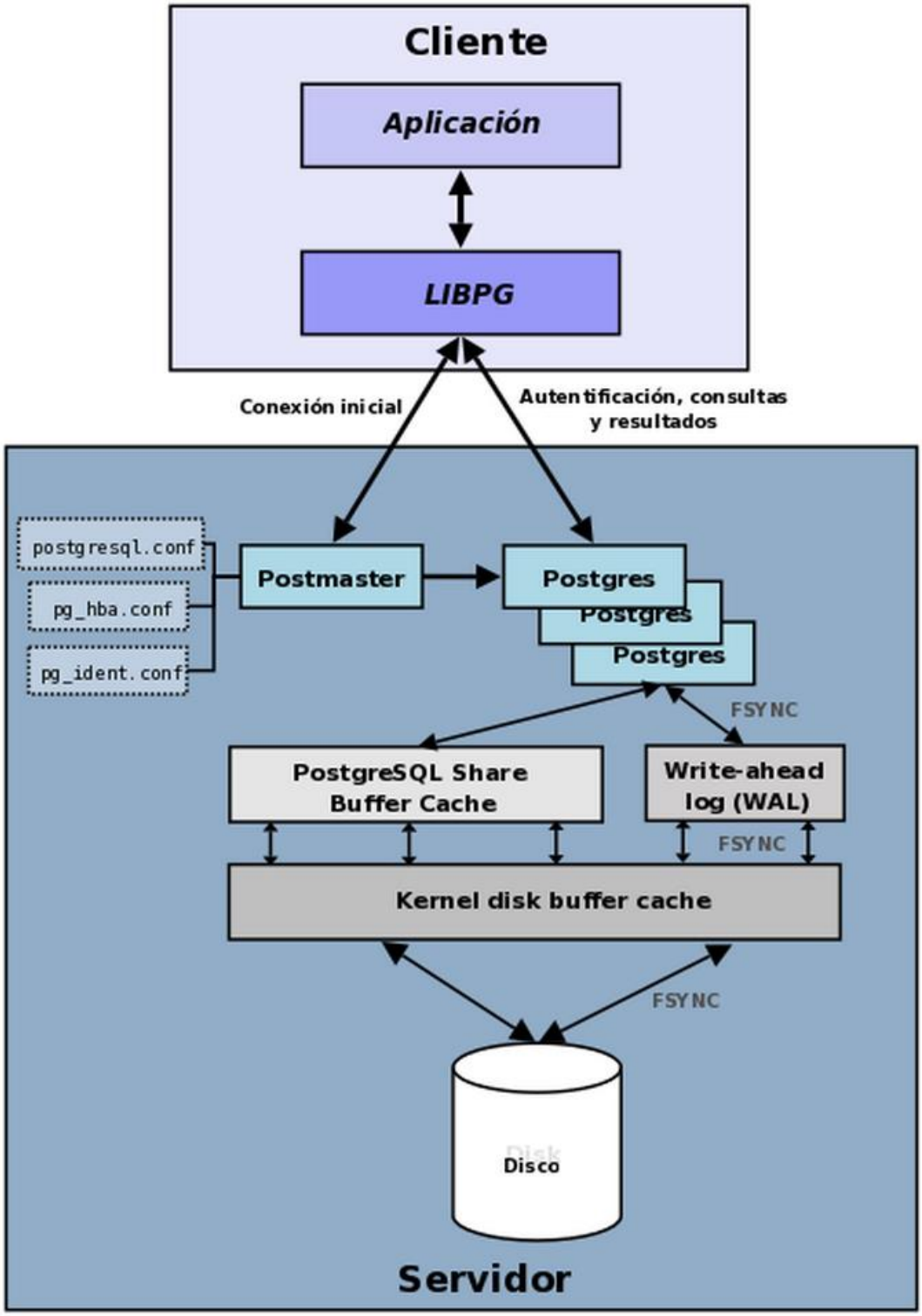

Figura 3.2. Componentes más importantes de un sistema PostgreSQL.[12]

[12] http://www.postgresql.org.es/sobre_postgresql

Características.

Sus características técnicas la hacen una de las bases de datos más potentes y robustas del mercado. Su desarrollo comenzó hace más de 16 años y, durante este tiempo, la estabilidad, potencia, robustez, facilidad de administración e implementación de estándares han sido las características que más se han tenido en cuenta durante su desarrollo. PostgreSQL funciona muy bien con grandes cantidades de datos y una alta concurrencia de usuarios accediendo al mismo tiempo al sistema.

Características generales.

- Es una base de datos 100% ACID (*Atomicity, Consistency, Isolation and Durability*).
- Integridad referencial.
- Transacciones anidadas.
- Replicación asíncrona/síncrona.
- PITR (*Point In Time Recovery*).
- Copias de seguridad "en vivo" (*Online hot backups*).
- *Unicode*.
- Juegos de caracteres internacionales.
- Regionalización por columna.
- MVCC (*Multi-Version Concurrency Control*).
- Múltiples métodos de autenticación.
- Acceso cifrado vía SSL.
- Actualización in-situ integrada (pg_upgrade).
- Completa documentación.
- Licencia BSD.
- Disponible para Linux y UNIX en todas sus variantes (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64) y Windows 32/64bit.

Características de programación y desarrollo.

- Funciones y procedimientos almacenados en numerosos lenguajes de programación, entre ellos PL/pgSQL (similar al PL/SQL de Oracle), PL/Perl, PL/Python y PL/Tcl.
- Bloques anónimos de código de procedimientos (sentencias DO).

- Numerosos tipos de datos y posibilidad de definir algunos nuevos. Además de los tipos estándares en cualquier base de datos, PostgreSQL tiene disponibles tipos geométricos, de direcciones de red, de cadenas binarias, UUID, XML, matrices, entre otros.
- Soporta el almacenamiento de objetos binarios grandes (gráficos, videos, sonido, y más).
- APIs para programar en C/C++, Java, .Net, Perl, Python, Ruby, Tcl, ODBC, PHP, Lisp, Scheme, Qt y muchos otros.

Características SQL.

- SQL92, SQL99, SQL2003, SQL2008.
- Llaves primarias y foráneas.
- Comprobación de restricciones únicas y no nulas.
- Restricciones de unicidad.
- Columnas auto-incrementales.
- Índices compuestos, únicos, parciales y funcionales en cualquiera de los métodos de almacenamiento disponibles: B-tree, R-tree, hash o GiST.
- *Sub-selects*.
- Consultas recursivas.
- Funciones "Windows".
- Joins.
- Vistas.
- Disparadores comunes, por columna y condicionales.
- Reglas.
- Herencia de tablas.
- Eventos LISTEN/NOTIFY.

3.10 DRBD (*Distributed Replicated Block Device*).

DRBD es una, poco común, solución de almacenamiento replicado basado en *software* que refleja el contenido de los dispositivos de bloque (discos duros, particiones, volúmenes lógicos, etcétera) entre los servidores.

DRBD replica la información con las siguientes características:

- En tiempo real. La replicación ocurre continuamente mientras las aplicaciones modifican la información en el dispositivo.
- Transparentemente. Las aplicaciones que almacenan su información en el dispositivo replicado son ajenas al hecho de que los datos en realidad están siendo almacenados en varias computadoras.
- Sincrónicamente o asincrónicamente. Con la replicación síncrona, una aplicación que almacena información es notificada sobre la finalización de la escritura sólo después de que ésta fue llevada a cabo en todas las computadoras espejo. La replicación asíncrona significa que la aplicación será notificada una vez que terminó el proceso localmente en el sistema, pero antes de que sea propagada a todos los demás sistemas replicados.

La funcionalidad del núcleo de DRBD es implementada por medio de un módulo del kernel de Linux. En concreto, DRBD constituye un controlador para un dispositivo de bloque virtual, por lo que DRBD está situado "muy cerca del fondo" de la pila de entradas y salidas del sistema. Debido a esto, DRBD es extremadamente versátil y flexible, características que lo hacen una solución de replicación de información adecuada para agregar alta disponibilidad a casi cualquier aplicación.

DRBD es, por definición y según el mandato de la arquitectura del kernel de Linux, agnóstico de las capas superiores. Esto significa que es imposible para DRBD agregar milagrosamente características a las capas superiores diferentes a las que ya poseen. Por ejemplo, DRBD no puede auto-detectar una corrupción en el sistema de archivos.

3.11 LVM (*Logical Volume Management*).

Logical Volume Management provee una vista de alto nivel de los discos de almacenamiento en un sistema computacional Linux. Esto proporciona al administrador del sistema mucha mayor flexibilidad en la asignación de volúmenes de almacenamiento para aplicaciones y usuarios. Consecuentemente, LVM otorga un mayor control y una mejor consulta de la información de almacenamiento que las vistas tradicionales de discos y particiones ofrecidas por los sistemas operativos.

Los volúmenes de almacenamiento creados bajo el control de LVM pueden ser redimensionados o movidos casi a voluntad, aunque esto puede requerir de la actualización de algunas herramientas del sistema.

LVM también hace posible la administración de volúmenes de almacenamiento en grupos de usuarios definidos que permiten al administrador del sistema trabajar con nombres sensiblemente más sencillos, tales como "desarrollo" y "ventas", en lugar de nombres de dispositivos físicos como "sda" y "sdb".

La administración de volúmenes lógicos se asocia tradicionalmente con largas instalaciones que contienen muchos discos físicos de almacenamiento, pero es igualmente adecuada para pequeños sistemas con un único disco o tal vez dos.

Los beneficios de LVM en redes pequeñas evidentemente se notan en usuarios con estaciones de trabajo caseras o en oficinas donde no son necesarias arquitecturas más complejas de almacenamiento, más allá de una PC con un disco duro local.

Las ventajas de LVM en sistemas amplios con varios discos de almacenamiento son mucho más obvias. La administración de grandes arreglos de discos es un trabajo que demanda mucho tiempo, volviéndose particularmente compleja si el sistema contiene discos de diferentes tamaños. El balanceo de almacenamiento entre varios usuarios puede ser muy conflictivo.

Los grupos de usuarios pueden ser asignados a grupos de volúmenes y volúmenes lógicos; y eso puede crecer tanto como se requiera. Es posible para un administrador del sistema "contener" o dejar fuera a un disco de almacenamiento hasta que sea requerido. Una vez que se necesite, puede ser agregado al grupo de volúmenes que tenga la necesidad más apremiante.

Cuando nuevos discos son incluidos en el sistema, ya no es necesario mover los archivos de los usuarios para hacer el mejor uso de la nueva disposición de almacenamiento; simplemente se agrega el nuevo disco a un grupo (o grupos) de volúmenes existente y se extienden los volúmenes lógicos como sea necesario.

Con LVM es también muy sencillo tomar unidades antiguas que están fuera de servicio y mover los datos que almacenan hacia nuevos discos. Esto puede hacerse con el sistema en línea, sin interrumpir el servicio a los usuarios.

La Figura 3.3 muestra la anatomía general de una arquitectura construida con LVM.

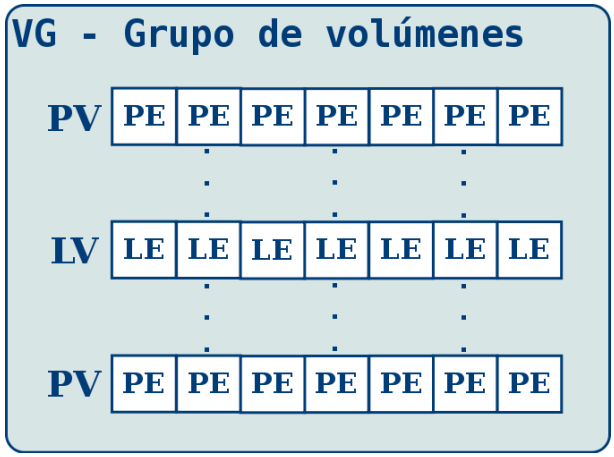

Figura 3.3. Arquitectura de elementos que define LVM.

A continuación una breve explicación de los elementos que componen la arquitectura de la Figura 3.3:

- VG – Grupo de Volúmenes. Es el nivel más alto de abstracción usado dentro de LVM. Conjunta una colección de volúmenes lógicos y volúmenes físicos en una misma unidad administrativa.
- PV – Volumen Físico. Es típicamente un disco duro. Aunque bien puede ser simplemente un dispositivo que "se ve" como un disco duro (un RAID, por ejemplo).
- LV – Volumen Lógico. Es el equivalente a una partición en un sistema tradicional que no es LVM. Un LV es visible como un dispositivo de bloques estándar; como tal, el Volumen Lógico puede contener un sistema de archivos (/home, por ejemplo)
- PE – Extensión Física. Cada volumen físico se divide piezas de datos, conocidos como extensiones físicas. Estas extensiones tienen el mismo tamaño que las extensiones lógicas para el grupo de volúmenes.
- LE – Extensión Lógica. Cada volumen lógico se divide en fragmentos de datos, conocidos como extensiones lógicas. El tamaño de la extensión es el mismo para todos los volúmenes lógicos del grupo de volúmenes.

Un ejemplo muy concreto de un sistema de almacenamiento administrado con LVM es el siguiente:

Supongamos que tenemos un grupo de volúmenes llamado VG1. Este grupo de volúmenes tiene una extensión física de tamaño 4 [MB]. Dentro de este grupo de volúmenes introducimos 2 particiones del disco duro: /dev/hda1 y /dev/hdb1. Dichas particiones se convertirán en los volúmenes físicos PV1 y PV2 (recuerde que los administradores pueden otorgar nombres sensiblemente más significativos). Los PV se dividen en trozos de 4 [MB], ya que este es el tamaño de la extensión para el grupo de volúmenes. Los discos son de diferentes tamaños y tendremos 99 extensiones en PV1 y 248 en PV2.

Ahora podemos crear nosotros mismos un volumen lógico que puede ser de cualquier tamaño entre 1 y 347 (resultado de sumar 248+99) extensiones. Cuando se crea el volumen lógico se define un mapeado entre extensiones lógicas y físicas. Por ejemplo: la extensión lógica 1 se podría asignar en la extensión física 51 de PV1, es decir, los datos escritos en los primeros 4 [MB] del volumen lógico serían puestos en la extensión 51 de PV1.

Capítulo 4.

Diseño e implementación.

Conociendo las herramientas de *software* que permitieron desarrollar y sustentar la viabilidad del proyecto, en el presente capítulo se abordará el tema del *hardware* utilizado, así como una explicación sobre el diseño y la implantación de la solución.

4.1 Recursos.

Para poder listar con mejor orden los recursos disponibles para el proyecto, es necesario dividirlos en tres categorías: infraestructura, servicios y *hardware* disponible.

Infraestructura.

La compañía tiene ubicadas sus instalaciones al Sur de la Ciudad de México, lugar donde se lleva a cabo el desarrollo del *hardware* y *software*, así como todas las actividades propias del NOC (*Network Operations Center*) para monitorear y operar de los dispositivos M2M remotos. Dentro de las instalaciones se tiene un *site* de comunicaciones –al que llamaremos "*site* 1"– que dispone de los elementos y servicios suficientes para dar cabida y operación a cualquier tipo de proyecto que los clientes demanden.

Además de las instalaciones principales, existe un contrato con un proveedor de *hosting* y servicios informáticos y de comunicaciones con sede en San Antonio, Texas. Dicho proveedor es una empresa con presencia a nivel mundial y es reconocida por la calidad de los servicios que proporciona a sus clientes. Con él se tienen hospedados en la ciudad de Chicago, Illinois, tanto los *routers* que harán posible la lógica de comunicación WAN, como los servidores web, de base de datos y de almacenamiento masivo en red; así como múltiples salidas a Internet con diferentes ISP que se respaldan entre sí para garantizar una disponibilidad de conexión del 100%. Toda esa infraestructura constituye el *site* 2.

Servicios.

En el *site* 1 se contrató a Telmex un enlace dedicado simétrico de 2 [Mbps] por *Frame Relay* desde nuestras oficinas hasta las instalaciones del *carrier*, con sede también en la Ciudad de México.

Además del anterior, se utilizó un enlace dedicado a Internet por microondas, contratado a la compañía Maxcom, con 10 [Mbps] de *uplink* y la misma capacidad en *downlink*.

Se incluyó una tercera conexión simétrica a Internet a través de la red de Alestra, con capacidad de 10 [Mbps].

El acceso a Internet desde el *site* en Estados Unidos tiene un ancho de banda simétrico de 100 [Mbps].

Hardware.

Dentro de los dispositivos disponibles en el *site* de la Ciudad de México, se utilizaron los mencionados en la siguiente lista, con sus respectivas características.

- 2 *routers* Acrosser AR-R5800; procesador Intel Core 2 Duo Quad Q9400 de 64 bits a 2.66 [GHz]; disco duro de 250 [GB]; 4 [GB] en RAM DDR3; 8 puertos Ethernet; Sistema Operativo Linux Debian 6.0.10 Squeeze.
- Cisco ISR modelo 1921 con IOS versión 15.2(4)M3.
- 1 *switch* Netgear FS728TP Smart.

Los elementos disponibles en el *site* de Chicago son:

- 2 *firewalls* Cisco ASA 5510 Sec+. Uno activo y otro de respaldo.
- 2 balanceadores de carga Brocade ADX 1000 Series. Uno activo y otro de respaldo.
- 2 servidores DELL PowerEdge R710; cada uno con dos procesadores Single Socket Six Core Intel Xeon E5645 de 64 bits a 2.4 [GHz]; 2 discos duros de 300 [GB] a 15000 RPM, configurados para replicarse con RAID 1; 12 [GB] en RAM DDR3; Sistema Operativo Linux Debian 6.0.10 Squeeze.
  Estos servidores se utilizaron para el servicio web.
- 2 servidores DELL PowerEdge R710; cada uno con dos procesadores Single *Socket* Six Core Intel Xeon E5645 de 64 bits a 2.4 [GHz]; 2 discos duros de 500 [GB] a 15000 RPM, configurados para replicarse con RAID 1; 24 [GB] en RAM DDR3; Sistema Operativo Linux Debian 6.0.10 Squeeze.
  Estos servidores se utilizaron para el servicio de base de datos.
- Un LUN (*Logical Unit Number*) de 1 [TB] en una SAN (*Storage Area Network*) para almacenamiento de la información de base de datos.
- 2 *routers* Cisco ISR modelo 1921; Sistema Operativo IOS versión 15.2(4)M4. Uno activo y otro de respaldo.

Evidentemente hay varios elementos más involucrados en ambos sitios, tales como *switches*, *bridges*, servidores de otros recursos, baterías de respaldo, NTU, módems, *firewalls*, etcétera. Sin embargo, no se mencionan como elementos clave en el diseño del proyecto debido a que forman parte de la infraestructura de la red completa de la empresa.

La razón por la cual se tomó la decisión de diseñar un sistema de comunicaciones heterogéneo, utilizando elementos de diferente fabricante y sistemas operativos como Cisco y Linux fue puramente comercial, por cuestiones de niveles de atención y garantía de servicio por parte del proveedor de Estados Unidos. En la Ciudad de México ya se contaba con los dispositivos mencionados desde el momento en que el proyecto fue planteado, así que una de las primeras tareas fue hacer la investigación y las pruebas de compatibilidad suficientes para asegurarse de que la solución funcionaría.

4.2 Planeación.

El proyecto estuvo enfocado en mejorar y crecer un sistema de supervisión y comunicación con dispositivos M2M dispersos a lo largo y ancho de la República Mexicana. Los dispositivos deben estar permanentemente conectados a una red celular, de manera similar a la navegación por Internet que hacen los teléfonos celulares de segunda y tercera generación. El trabajo se centra en los servicios de comunicaciones que se prestan a través de un solo *carrier* de telefonía móvil con sede en México, ya que ese mismo esquema puede ser trasladado fácilmente a otro proveedor tanto en México como en otros países.

Naturalmente, al transmitir datos desde un dispositivo conectado a una red celular hacia un sitio lejano que pertenece a una red de otra organización, se deben buscar los medios para interconectar las dos redes involucradas y permitir que los paquetes de información fluyan entre ambas.

Por otro lado, una vez que la parte de comunicaciones esté resuelta, se tiene que garantizar el correcto procesamiento y almacenamiento de la información para que ésta pueda ser consultada por los usuarios que participan activamente en la vigilancia de los dispositivos.

En resumen, el diseño del proyecto puede dividirse en tres etapas:

i. Conectividad redundante entre un *carrier* de comunicaciones y el NOC de la compañía.
ii. Redundancia WAN entre el *carrier* y el NOC a través de un tercer sitio geográfico.
iii. Alta disponibilidad de los servidores mediante *clusters* de computadoras.

Los puntos mencionados serán abordados en ese orden a lo largo de la siguiente sección.

4.3 Diseño.

Probablemente la parte más delicada de un proyecto llega una vez que se aterrizan las ideas y tienen que traducirse en herramientas y componentes que permitan materializarlas. Es aquí, en el diseño, donde la capacidad y el conocimiento de los involucrados tienen que explotarse al máximo con la finalidad de crear un sistema robusto, a prueba de todos los potenciales inconvenientes y que además sea escalable, ofreciendo la posibilidad de crecer tanto como sea posible con el mínimo de cambios, inversión y configuraciones extras.

Tal como se mencionó en el capítulo 2, la forma tradicional de interconectar las redes de dos compañías es por medio de un enlace dedicado que generalmente se extiende por la red de un tercero, el cual garantiza ciertos niveles de seguridad, disponibilidad, confiabilidad y ancho de banda, dependiendo de la tecnología que utilice. Podemos considerar la instalación del enlace dedicado como el primer paso del diseño. Una vez que los paquetes de datos pueden llegar hasta la red corporativa del *carrier*, éste internamente se encarga de trasladarlos hasta su red celular y posteriormente entregarlos al dispositivo móvil. Ver Figura 4.1.

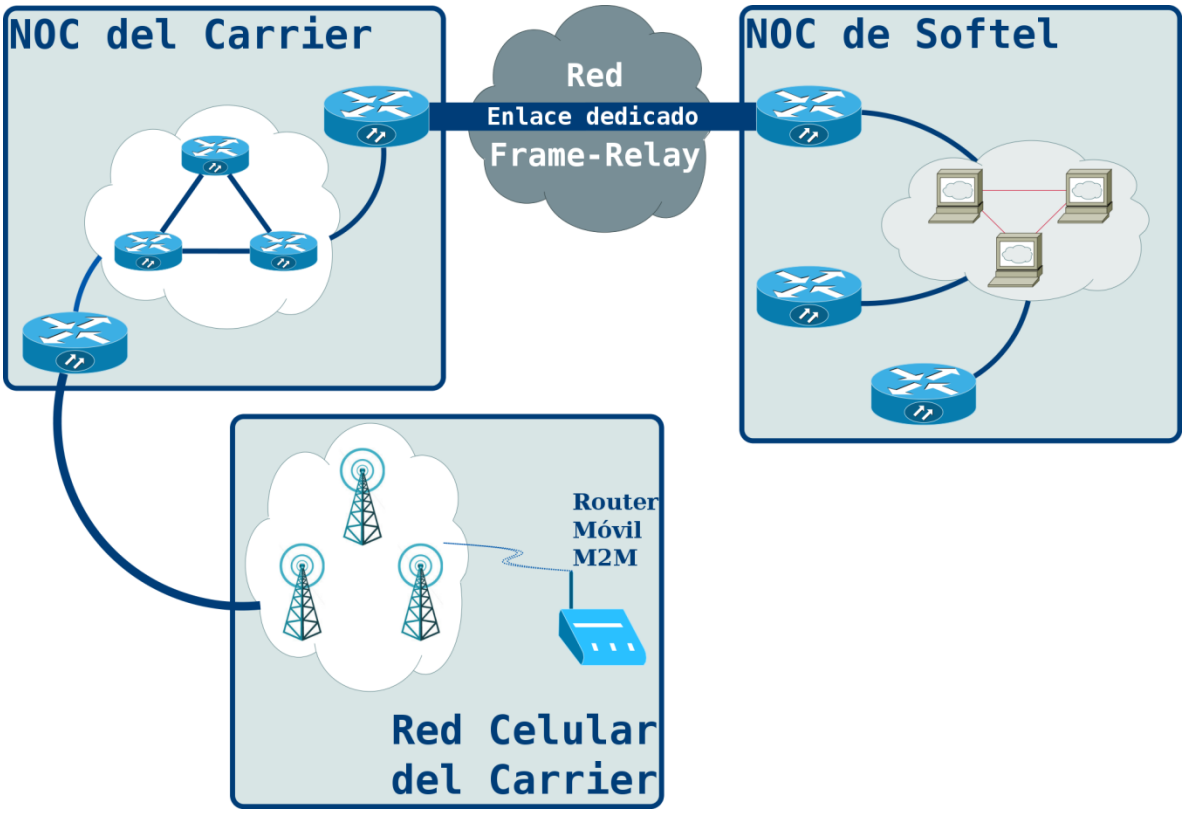

Figura 4.1. Enlace dedicado que conecta físicamente a dos *sites*.

El Cisco ISR 1921 es el encargado de recibir, en su interfaz Serial0/0/0, los datos provenientes del *carrier* a través del enlace dedicado en la Ciudad de México. La razón de haberlo elegido fue la facilidad de agregar la HWIC (*High-Speed WAN Interface Card*) con un puerto V.35. Esta infraestructura ya se tenía inicialmente y a través de este enlace era como se alcanzaban los segmentos de red celular asignados para los SIM desde el NOC de La Empresa.

Para robustecer la capacidad de comunicación con los dispositivos M2M y estar prevenidos ante cualquier anomalía en la red *Frame Relay* del enlace, se decidió agregar redundancia al enlace por medio de una VPN a través de Internet que funcione como medio alternativo. La redundancia también pudo haber sido agregada con otro enlace dedicado de respaldo, con un proveedor distinto e incluso una tecnología de WAN diferente, como MPLS, pero eso incrementaría el costo y no es una inversión que favorezca la rentabilidad de la solución.

Por definición, una VPN que se forma con un túnel IPsec no permite la propagación de cualquier tipo de paquetes y el *routing* no puede ser manejado con libertad, ya que se definen dominios de cifrado que deben estar autorizados desde la negociación de las ISAKMP. Por tanto, no promueve la flexibilidad al tener que agregar dos políticas de cifrado por cada par de segmentos de red que se deseen conectar, una en cada sentido de la comunicación. Tampoco es deseable establecer solamente la VPN con un túnel IPsec, debido a la ausencia de rutas con las métricas adecuadas para elegir una trayectoria u otra (enlace dedicado o VPN por Internet), lo cual implica que la conmutación en caso de falla deberá ser reactiva y requiere de la intervención de un administrador para hacerla posible. Un dispositivo con un túnel IPsec simple configurado para alcanzar determinada red, le dará prioridad a dicho túnel mientras la configuración para el mismo exista y sea válida; y no admitirá una ruta alterna a menos que parte de esa configuración sea borrada. Esto se convierte en un problema de automatización en el proceso de *failover*.

Hasta ahora hemos visto que los protocolos de *routing* dinámico son muy efectivos cuando se trata de compartir o eliminar rutas en función de los cambios que acontecen en la topología de una red, derivados de fallas o simplemente de modificaciones intencionales hechas por los propios administradores. El empleo de un protocolo de *routing* adecuado puede ayudar muchísimo en este caso; no obstante, en los túneles IPsec no se admite el cifrado de todos los paquetes involucrados en esa clase de protocolos, además de dar pie al mismo problema explicado anteriormente: se tienen las rutas hacia los segmentos de red privados, pero las políticas de seguridad no admitirán la comunicación entre ellos. Está la necesidad, entonces, de buscar un medio para lograr el intercambio de rutas con un protocolo y que además dicho intercambio vaya cifrado por el alto riesgo de inseguridad que existe en Internet, sin mencionar que la naturaleza de la propia Internet impide el enrutamiento de paquetes cuyas direcciones IP pertenecen a un segmento privado. En este caso, recordemos que las redes corporativas de las compañías generalmente “habitan” en los segmentos privados 10.0.0.0/8, 172.16.0.0/12 o 192.168.0.0/16.

Afortunadamente la solución está en un tema cubierto en el capítulo 2: los túneles GRE. El cifrado de paquetes GRE está permitido dentro de los túneles IPsec, mientras que GRE a su vez posibilita el encapsulamiento de muchos tipos de paquetes IP, incluyendo los pertenecientes a protocolos de *routing* dinámico. Las transmisiones de tipo *broadcast* y *multicast* a menudo son utilizadas por dichos protocolos.

Gracias a que GRE permite agregar, relativamente, tantas rutas como sean necesarias asociándolas a la interfaz de su correspondiente túnel sin alterar la configuración de seguridad del túnel IPsec, la solución se vuelve bastante flexible y escalable si además se incluye un protocolo de *routing* para automatizar la administración de las trayectorias que seguirán los paquetes. BGP es ideal para la solución por tratarse de un protocolo EGP, puesto que están involucradas dos redes que pertenecen a distintos Sistemas Autónomos, además de ser muy estable y eficiente.

Llega ahora el momento de concentrar todo lo descrito en los últimos párrafos. ¿Cómo combinar las características de un túnel IPsec, un túnel GRE y un protocolo de *routing* en una sola configuración? Resulta evidente que el cifrado de los paquetes debe ser la "capa" más externa en la línea de comunicación que existe entre los *routers* de borde, por lo tanto el túnel IPsec es quien debe contener absolutamente toda la información que se transmite en el medio. Para ello se tuvo que establecer una VPN que no fue *site-to-site*, sino *host-to-host*. ¿Cuál es la diferencia? Simple, una VPN *site-to-site* conecta todo un segmento de red de una compañía con otro segmento de red de otra compañía; para efectos más prácticos, léase "todo el departamento de finanzas de una sucursal, con el departamento de RH de otra sucursal", por ejemplo. Por otro lado, la VPN *host-to-host*, como su nombre lo indica, permite la comunicación entre un solo nodo (servidor, *router* o usuario) de un lado contra un solo nodo que "vive" detrás del otro extremo.

La razón por la cual nos interesa conectar sólo un nodo por bando con la VPN tiene que ver con la propia definición de túnel en redes: un enlace virtual punto a punto. De tal forma que los dominios de cifrado de la VPN serán las puntas entre las que se levante el túnel GRE, como se muestra en la Figura 4.2.

Una vez que se planearon las herramientas que permiten comunicar diferentes elementos entre dos redes privadas, vamos a ver los requisitos para montar un protocolo de *routing* entre dos *routers*. El primero de ellos es que ambos deben ser alcanzables entre sí, ya sea porque pertenecen a la misma red o por medio de una ruta establecida por el administrador. Cualquiera de los dos casos es aceptable para BGP.

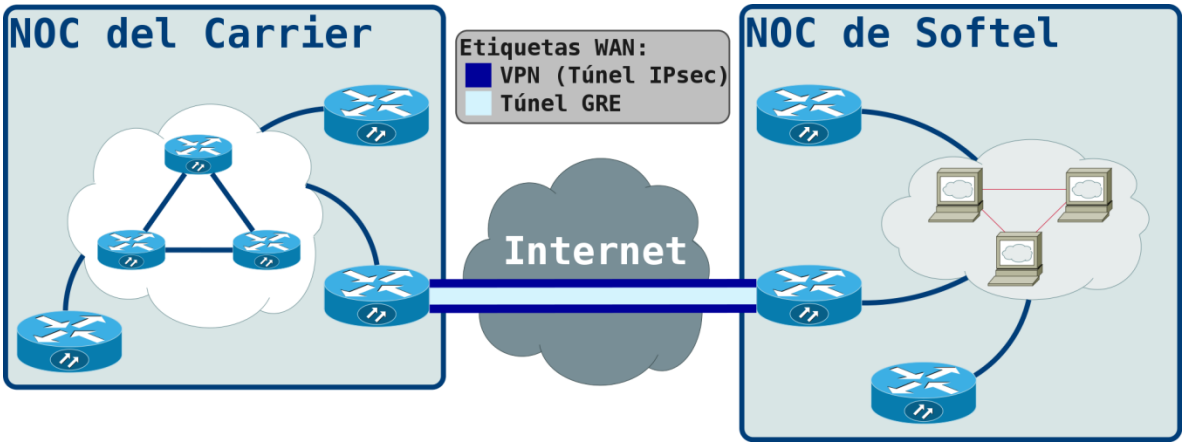

Figura 4.2. VPN que conecta lógicamente
a dos *sites* usando un túnel GRE.

BGP necesita de algunas características que tomamos en cuenta inicialmente. Necesitamos un identificador para nuestro propio Sistema Autónomo dentro del segmento privado, así como las direcciones que tendrá cada *router* para formar la vecindad entre ellos. La opción más recomendable fue que ambos formen parte de un mismo segmento de red IP. Una red con una máscara de 30 bits de longitud sería suficiente, ya que en ella hay dos direcciones de *host* disponibles, además de la dirección propia de la red y la de *broadcast*.

Una vez más, utilizar un túnel GRE facilita la solución; finalmente representa una interfaz virtual que puede ser configurable en el *router* y dentro de esas opciones configurables se encuentra el direccionamiento administrativo. Si a la interfaz del túnel se le asigna una dirección IP dentro de una red, automáticamente todos los paquetes destinados a otra dirección que se encuentre dentro de esa misma red serán encapsulados con GRE y enviados hacia el *peer* del otro lado del túnel, quien hará el proceso inverso al encapsulamiento y decidirá qué hacer con ellos. Si el *peer* pertenece a la misma red y los paquetes van dirigidos precisamente a él, habrá comunicación directa entre los *routers* por "dentro" del túnel GRE. Así, la vecindad de BGP puede ser levantada entre esas dos direcciones y el proceso de cifrado, encapsulamiento e intercambio de información de *routing* tendrá lugar en el mismo dispositivo (como lo muestra la Figura 4.3), pero en diferentes niveles de procesamiento y con un orden determinado.

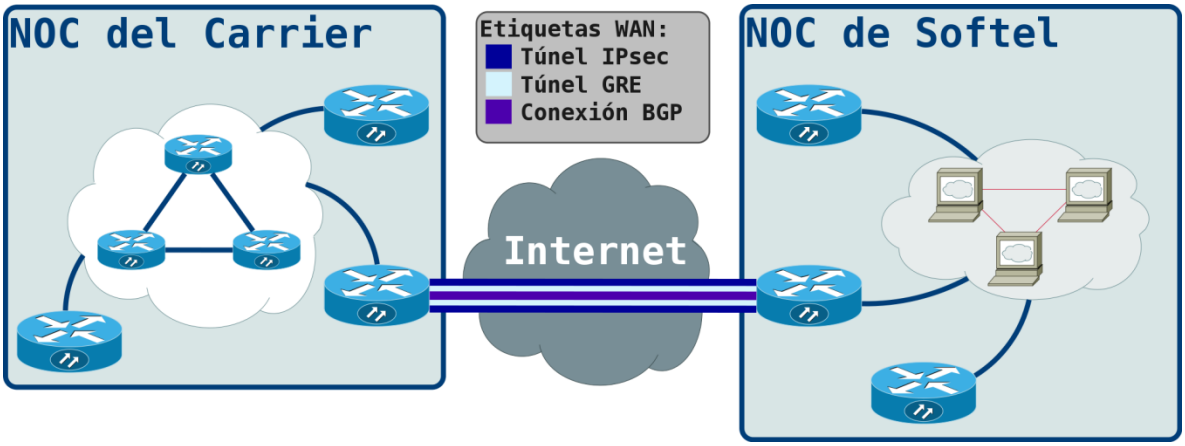

Figura 4.3. Sesión BGP entre dos *sites* a través de una VPN con túnel GRE.

El caso del enlace dedicado fue más sencillo, porque *Frame Relay* permite el intercambio de mensajes de BGP a través de los circuitos virtuales de su red WAN que se encuentran, de facto, direccionados dentro de la misma red IP (véase Figura 4.4). Por lo tanto, el establecimiento de la vecindad de BGP se dio de forma natural.

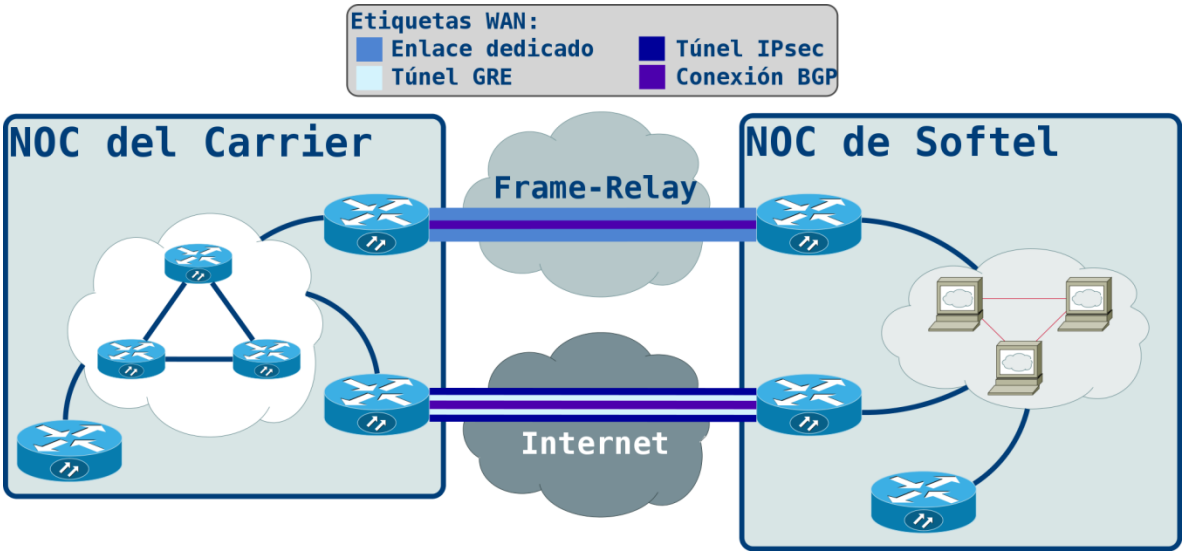

Figura 4.4. Redundancia de BGP entre dos *sites* por medio de un enlace dedicado y una VPN con túnel GRE.

En un esquema como el anterior se tiene cubierta la parte de redundancia entre el *carrier* y el *site* de la Ciudad de México. El siguiente paso fue incluir el *site* de Chicago para agregar redundancia geográfica al sistema, formando un triángulo desde el punto de vista gráfico.

Respecto a las ventajas que ofrece el proveedor de Estados Unidos, recordemos que una de ellas es la garantía de disponibilidad en Internet del 100%. Por ese lado, podemos estar tranquilos sabiendo que no es necesaria una configuración de doble enlace, tal como se planeó para la Ciudad de México, en donde los enlaces están garantizados por contrato en un 99.8% y no se descartan períodos de intermitencia en el servicio durante el transcurso del año. Derivado de ello, se acordó también tener un par de túneles IPsec, cada uno por diferente proveedor, hacia una sola dirección IP pública en Chicago. De esa forma, si un proveedor de Internet en la Ciudad de México falla y esa VPN queda fuera de línea, aún se tiene la otra para seguir dando servicio tanto a los clientes como al personal de la compañía que labora en México. Cabe destacar que uno de los *routers* utilizados para conectar con el *site* 2 es el mismo que el empleado para la VPN con el *carrier*. Con eso se evita agregar más puntos de falla al diseño y se economiza en la disposición de *hardware*, energía y renta de servicios.

Se asignó un Sistema Autónomo distinto al *site* de Chicago con el objetivo de hacer más sencilla la configuración de BGP en el escenario final, ofreciendo una lógica más "natural" al momento de definir las prioridades de las rutas, situación que también se traslada hasta el *carrier*.

La arquitectura de redundancia entre el NOC y Chicago es prácticamente idéntica a la que se planeó del NOC al *carrier*, con un par de VPN *host-to-host* y túneles GRE con una sesión BGP dentro de cada uno, cuya vecindad se forma con las direcciones IP en las interfaces virtuales de los túneles usando máscaras de 30 bits. Ver Figura 4.5.

El diseño de la comunicación entre el *carrier* y el *site* 2 es mucho más sencillo, gracias a que las direcciones IP públicas de los *firewalls* en cada sitio tienen una disponibilidad garantizada permanentemente, por lo que sólo fue necesaria una VPN con la misma estructura que las demás: un túnel IPsec con un túnel GRE, más una sesión BGP.

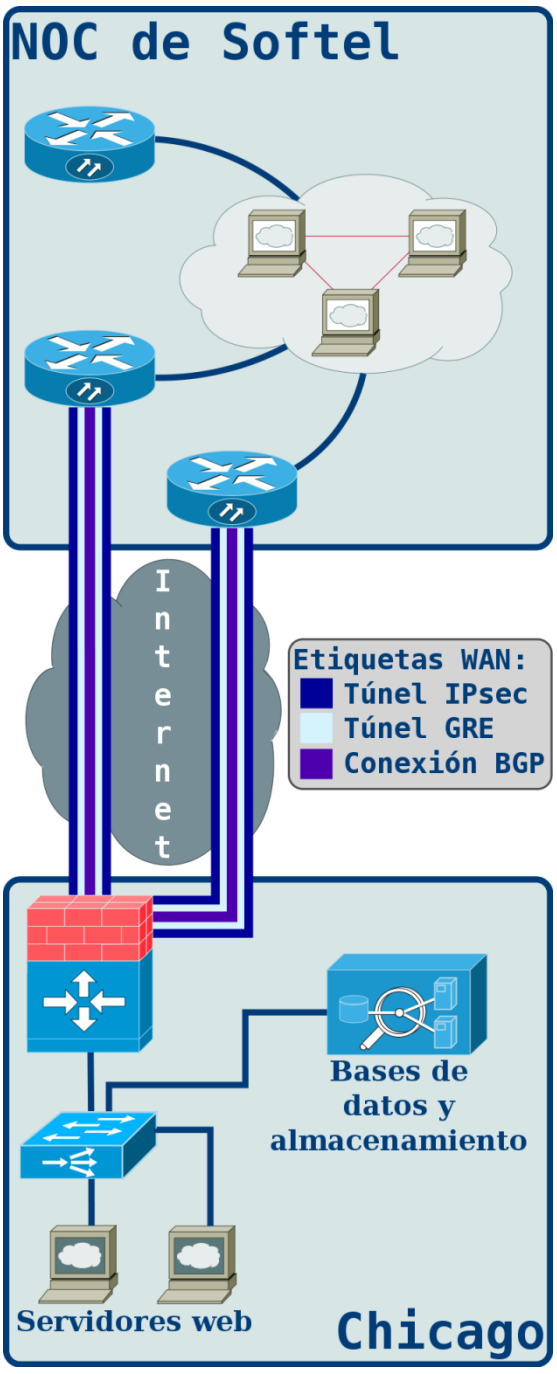

Figura 4.5. Redundancia de BGP entre dos *sites* por medio de dos VPN con túnel GRE.

Armando cada uno de los bloques descritos en los párrafos de esta sección, se puede presentar el esquema completo de conectividad en la Figura 4.6 con el triángulo de redundancia ya mencionado.

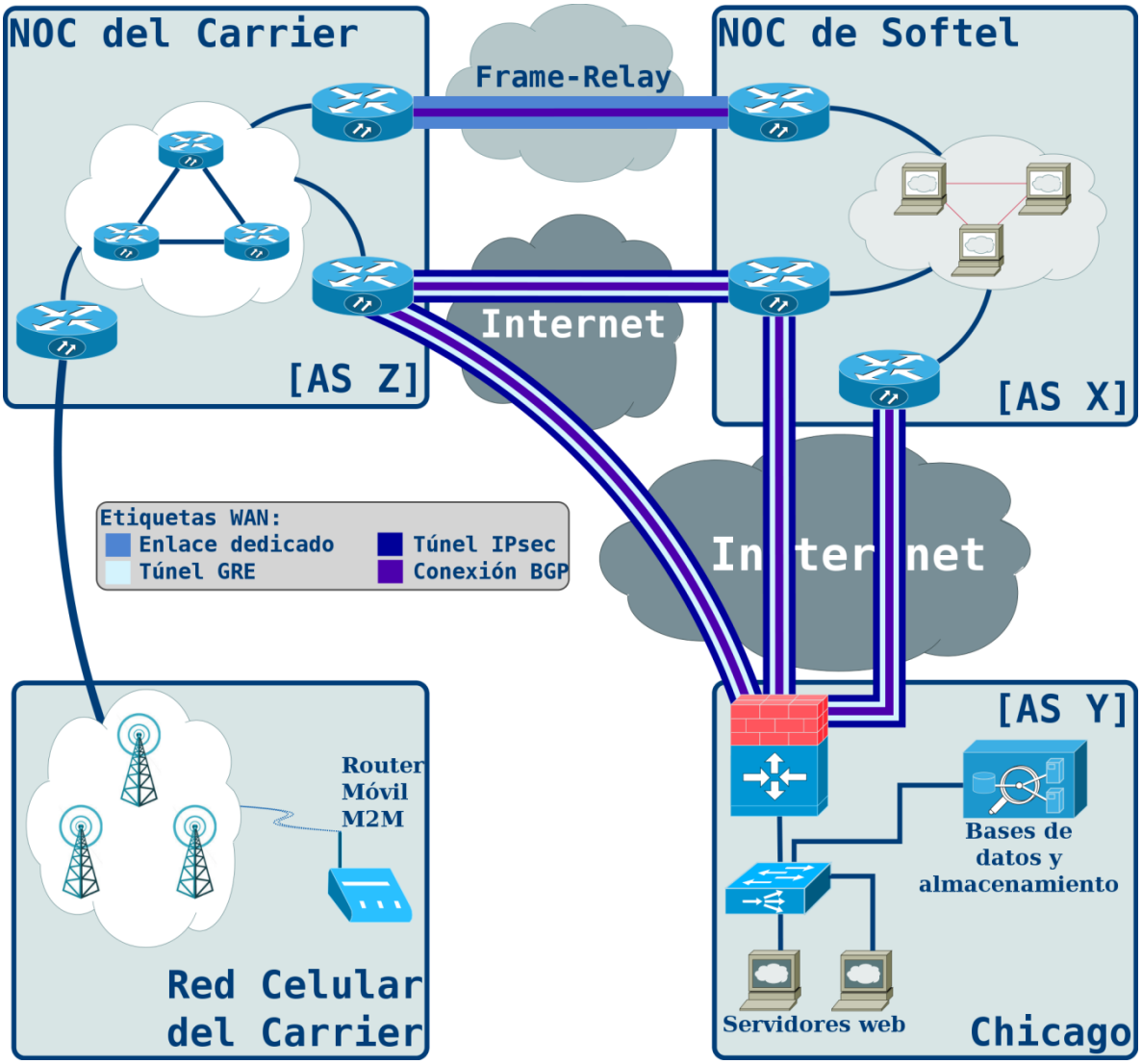

Figura 4.6. Triángulo de redundancia
entre tres *sites* con BGP.

El diseño anterior posibilita las características listadas después de la Figura 4.7, en donde los enlaces WAN se han simplificado para un mejor análisis.

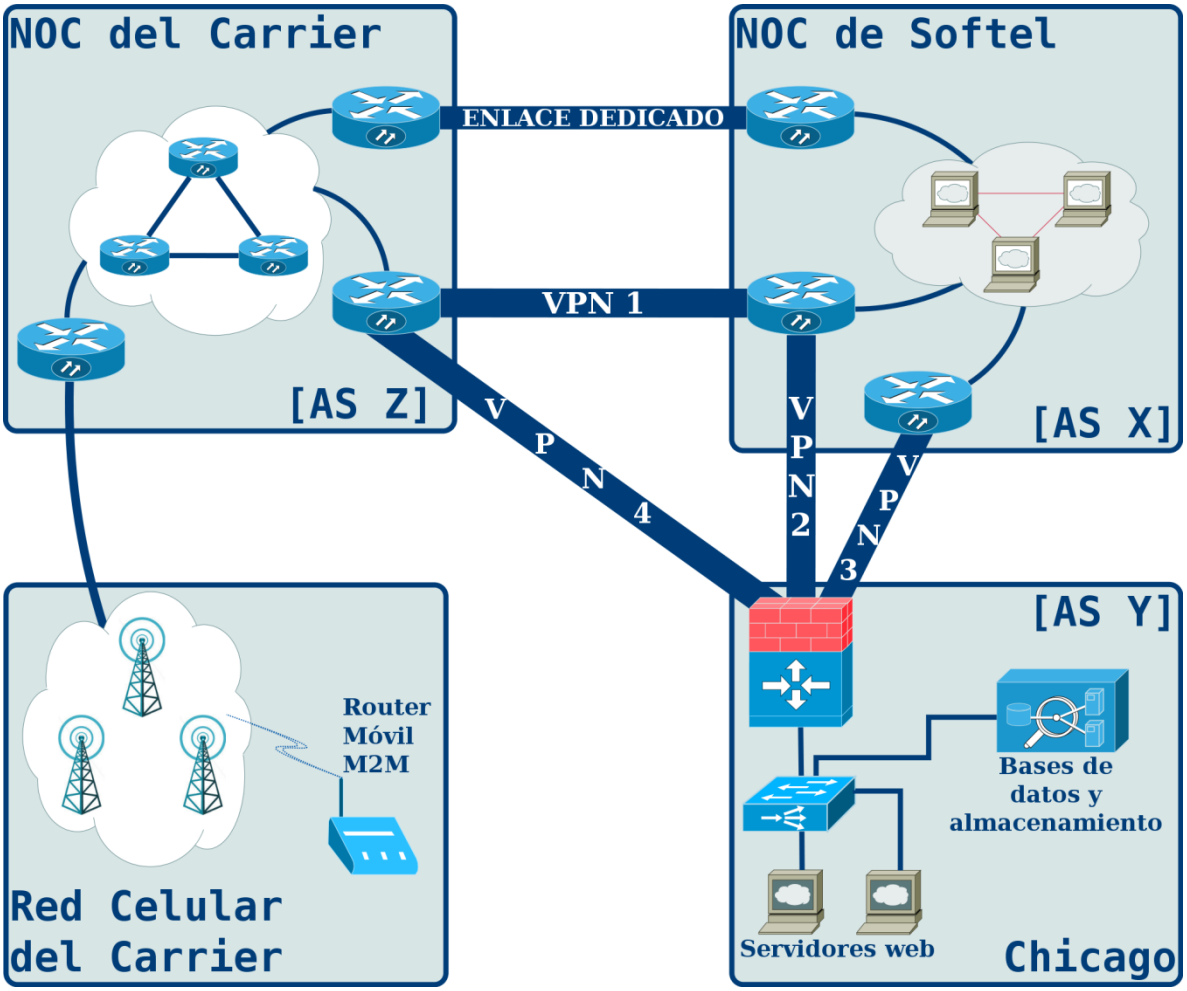

Figura 4.7. Triángulo de redundancia
entre tres *sites*, simplificado.

- El *site* 1 tiene comunicación con la red celular directamente por el *carrier* a través de dos enlaces posibles (enlace dedicado y VPN1).
- Desde el *site* 1 también se puede llegar a los móviles a través del *site* 2, por dos trayectorias disponibles (VPN2 o VPN3 y posteriormente VPN4).
- El *site* 2 alcanza el segmento de móviles directamente por la VPN4.
- Desde el *site* 2 se llega también a la red celular a través del *site* 1, por la VPN2 o VPN3 (y luego por medio del enlace dedicado o la VPN1).
- Los dispositivos móviles pueden alcanzar tanto el *site* 1 como el *site* 2 por cualquiera de los medios y combinaciones de los puntos anteriores, con el flujo del tráfico en sentido inverso.

Ahora que el diseño general de redundancia en comunicaciones se ha desmenuzado, el siguiente paso es detallar el diseño para la redundancia de los servidores mediante la implantación de un *cluster*.

Tal como se explicó en el primer capítulo, un *cluster* permite a un conjunto de computadoras crecer la disponibilidad de uno o varios de sus recursos informáticos mediante el respaldo o la unificación de sus capacidades. En esta sección explicaré qué tipos de *cluster* se eligieron y cuál fue la razón para hacerlo.

Fueron dos servicios los de mayor interés para este proyecto: consultas por web y resguardo de información en base de datos. Ambos se deben ejecutar en un servidor independiente, con el respectivo respaldo de *hardware* para cada uno.

Apache2 fue el *software* utilizado para que los servidores web operen. Tradicionalmente es factible colocar a este tipo de servidores que ofertan las mismas plataformas dentro de un *cluster* de tipo activo-activo, con la finalidad de volver al sistema más eficiente y aligerar la carga de procesamiento en cada nodo. Pero dada la naturaleza de los procesos del sistema de monitoreo de La Empresa, inicialmente no fue posible hacerlo de esa forma y en primera instancia el *cluster* fue configurado para funcionar como activo-pasivo. Las razones por las cuales no fue posible adaptar el esquema salen del objetivo del reporte.

Respecto a los nodos de base de datos, éstos utilizan PostgreSQL para ofrecer el servicio. La información se queda localmente almacenada en un volumen lógico creado con LVM y replicado entre los nodos con ayuda de DRBD, que periódicamente guarda un respaldo de su información en el LUN de una SAN montada por red en un folder del servidor de base de datos. La SAN fue alquilada al proveedor de *hosting*, así que el mantenimiento y la disponibilidad en lo que a ella se refiere es responsabilidad del personal de Chicago y San Antonio.

El tipo de *cluster* seleccionado para la base de datos también es activo-pasivo y, a diferencia del *cluster* web, en este caso sí es el más recomendable por motivos de integridad y consistencia de los datos. No es deseable que dos o más servidores se encuentren manipulando la misma información en las bases de datos; al menos no al nivel de la complejidad que exige este proyecto.

Sobra decir que al ser ambos *clusters* activo-pasivo, el servicio de Apache2 sólo estará ejecutándose en un nodo y deberá estar completamente inactivo en el otro, a la espera de que el servicio falle para tomar su lugar. En el otro *cluster*, PostgreSQL también se estará ejecutando en un solo nodo y el volumen lógico de almacenamiento donde se guarda la información de las bases será montado en el mismo; si el nodo primario falla, el proceso de PostgreSQL será detenido y la carpeta se desmontará para evitar que los datos puedan alterarse, migrando todo completamente al nodo que permanecía pasivo hasta entonces.

4.4 Topología.

Partiendo de las bases establecidas para los *clusters*, voy ahora a describir el comportamiento del sistema completo en un estado normal y cuáles son los resultados que se esperan como respuesta a múltiples eventos. Esta vez, además de la topología y sus variantes, también iré aclarando ciertos puntos que se consideraron previos a la configuración y que tienen que ver con el *software* o el *hardware* utilizados.

La topología en la ciudad de Chicago no es tan sencilla como podría pensarse, a pesar de que el número de elementos de *hardware* no es alto. Al utilizar un Cisco ASA como *firewall* para construir las VPN, se pierde la posibilidad de crear un túnel GRE en el mismo *router* porque dicho protocolo es considerado como inseguro, mientras que los ASA son uno de los productos estrella de Cisco en materia de seguridad y su plataforma ni siquiera lo contempla. Además de lo anterior, dar el control del túnel GRE al proveedor de *hosting* quita a La Empresa toda la flexibilidad para agregar o quitar configuración conforme a sus necesidades.

Para solucionar el inconveniente, se creó una nueva zona llamada "segmento *routing*" en donde operan los Cisco ISR 1921, con dirección de red 172.16.206.0/27 e independiente a la LAN de La Empresa. Estos *routers* son los encargados de establecer el túnel GRE contra la dirección de *host* que funge como dominio de cifrado en el otro extremo de cada VPN.

Si se toma en cuenta que un túnel GRE simple sólo puede tener una punta en cada extremo, ¿por qué razón tener dos o más *routers* en el segmento *routing* y cómo decidir cuál es el que debe levantar el túnel? La respuesta es simple y tiene que ver con los objetivos principales que se buscaron solventar desde el inicio del proyecto: disponibilidad y redundancia. Uno de los Cisco ISR es el que se encuentra activo y operando todo el tiempo, mientras que el otro permanece a la espera de heredar sus funciones, si ocurre una falla.

La manera más sencilla y transparente que se encontró para hacerlo fue implementando VRRP entre ambos *routers*. La dirección IP del *router* virtual es 172.16.206.13 y esa es precisamente el *peer* del lado de Chicago para el túnel GRE, de tal manera que los dos Cisco ISR tienen exactamente la misma configuración (con excepción del nombre del nodo y la dirección IP de LAN) para que *failover* entre uno y otro no se note. Una vez que el *router* virtual recibe los paquetes con GRE, les quita el encapsulamiento y los regresa al *firewall* para que éste los encamine a su destino final en la LAN (Figura 4.8), que puede ser cualquiera de los servidores en la DMZ o los del segmento privado.

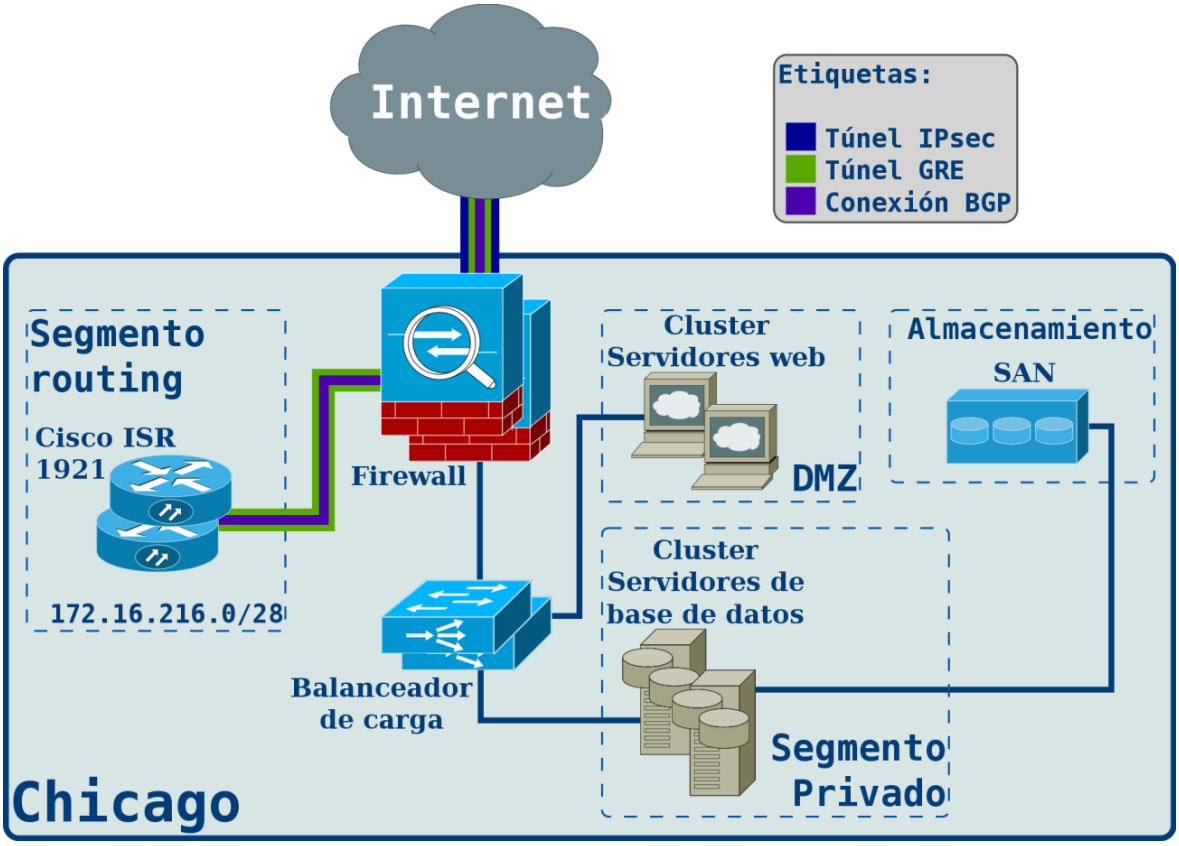

Figura 4.8. Diagrama de la red en Chicago.

Cabe aclarar que por cuestiones de administración de los *routers*, las VPN entre la Ciudad de México y Chicago son *host-to-site*, es decir, el dominio de cifrado en México es una sola dirección IP mientras que en los Estados Unidos es todo el segmento *routing* con máscara de 27 bits. En el caso de la VPN con el *carrier*, sí es *host-to-host* y el dominio de cifrado en el *site* 2 es la dirección IP del *router* virtual (172.16.206.13).

4.4.1 Redundancia WAN.

Teniendo claro cuántos son los enlaces y de qué manera se conectarán cada uno de los sitios, lo que resta es explicar el comportamiento que tienen y cuáles son las prioridades para comunicarse de un lugar a otro. Recordando un diagrama anterior y agregando algunos datos más específicos, se tiene la Figura 4.9.

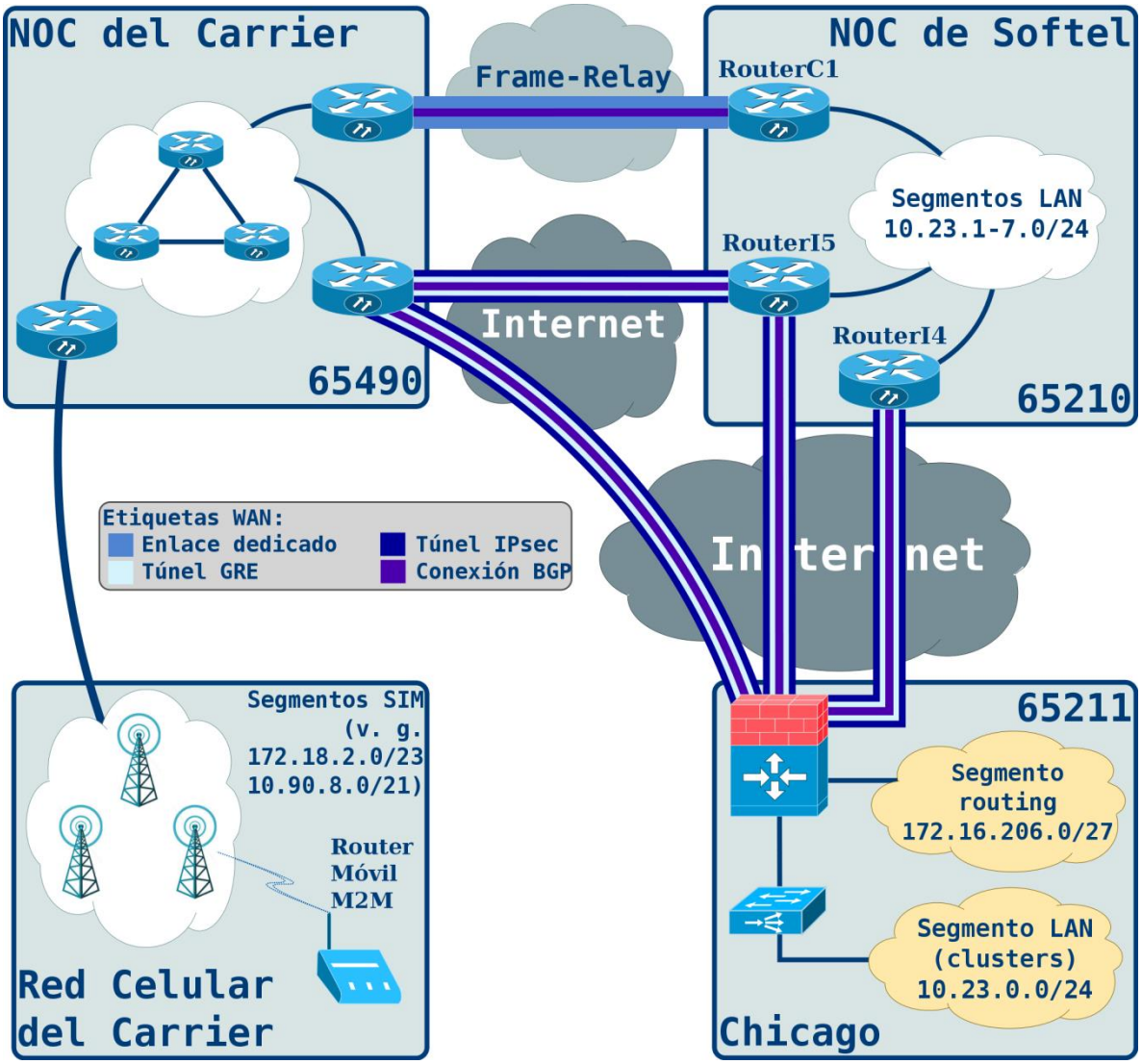

Figura 4.9. Triángulo de redundancia
entre tres *sites*, detallado.

La red de La Empresa se puede resumir en el segmento 10.23.0.0/21, pero por conveniencia se dividió en 8 segmentos con máscaras de 24 bits repartidos de la siguiente manera:

- La primera red 10.23.0.0/24 está ubicada sólo en el *site* de Chicago dentro del AS 65211.
- El resto de redes, desde 10.23.1.0/24 hasta 10.23.7.0/24, se utilizan o están reservadas de momento para el *site* de la Ciudad de México, que tiene el AS 65210.

El *carrier* permite llegar a dispositivos móviles con SIMs conectados a la red celular y direcciones IP que pertenecen a diferentes subredes a través del AS 65490. Por ejemplo: 10.90.8.0/21, 172.18.2.0/23.

Por requerimiento estricto del *carrier* y simplificación de la configuración para ellos, los enlaces deben tener una prioridad muy específica de acuerdo a la red que deben alcanzar, por lo tanto el balanceo de carga desde el *carrier* a La Empresa no está contemplado.

En ese sentido, las redes que se encuentran en la Ciudad de México tienen como medio principal −desde el *carrier*− al enlace dedicado, como enlace secundario la VPN1 y el medio terciario será la VPN4. Retomando el diagrama simplificado, pero con los datos de las subredes y los AS, la Figura 4.10 ilustra las prioridades del *carrier* para alcanzar los segmentos del *site* 1.

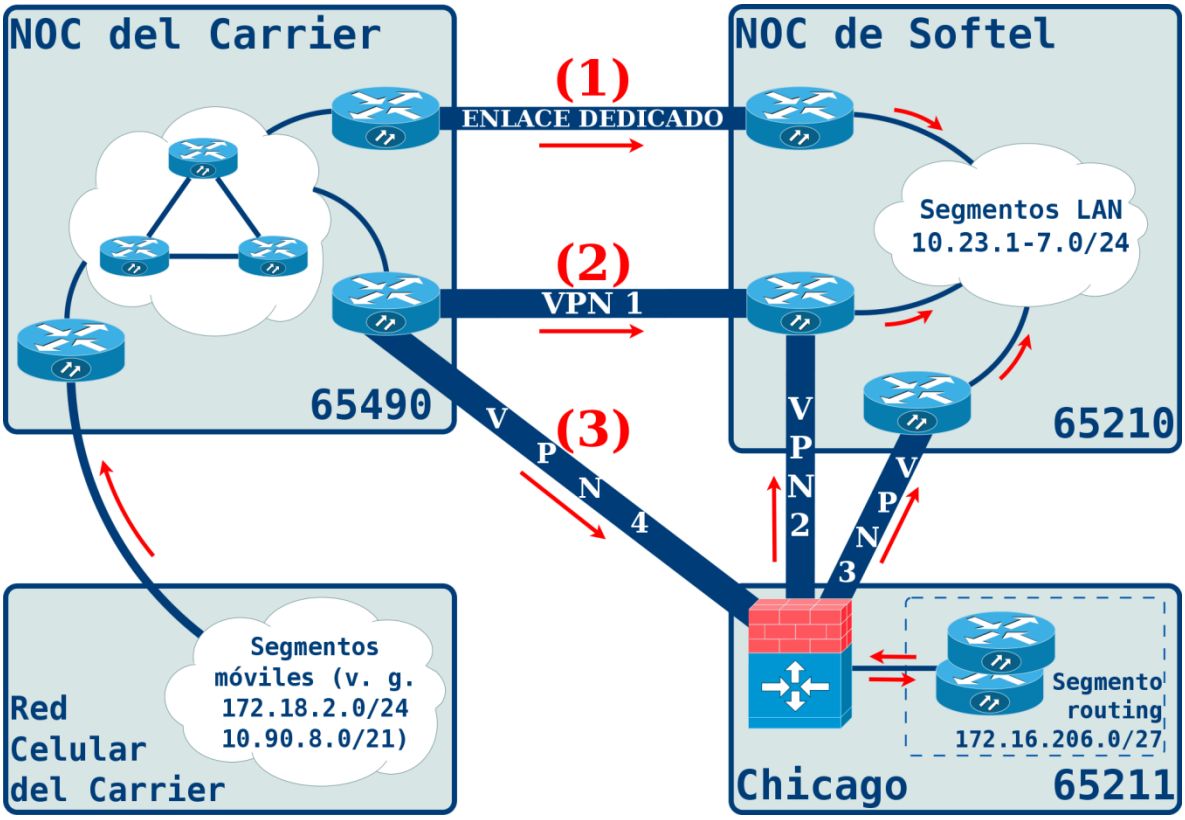

Figura 4.10. Posibles trayectorias
desde el *carrier* hacia el *site* 1.

Para llegar al segmento ubicado en Chicago, la Figura 4.11 muestra que la prioridad es la VPN4, seguida por el enlace dedicado y por último la VPN1.

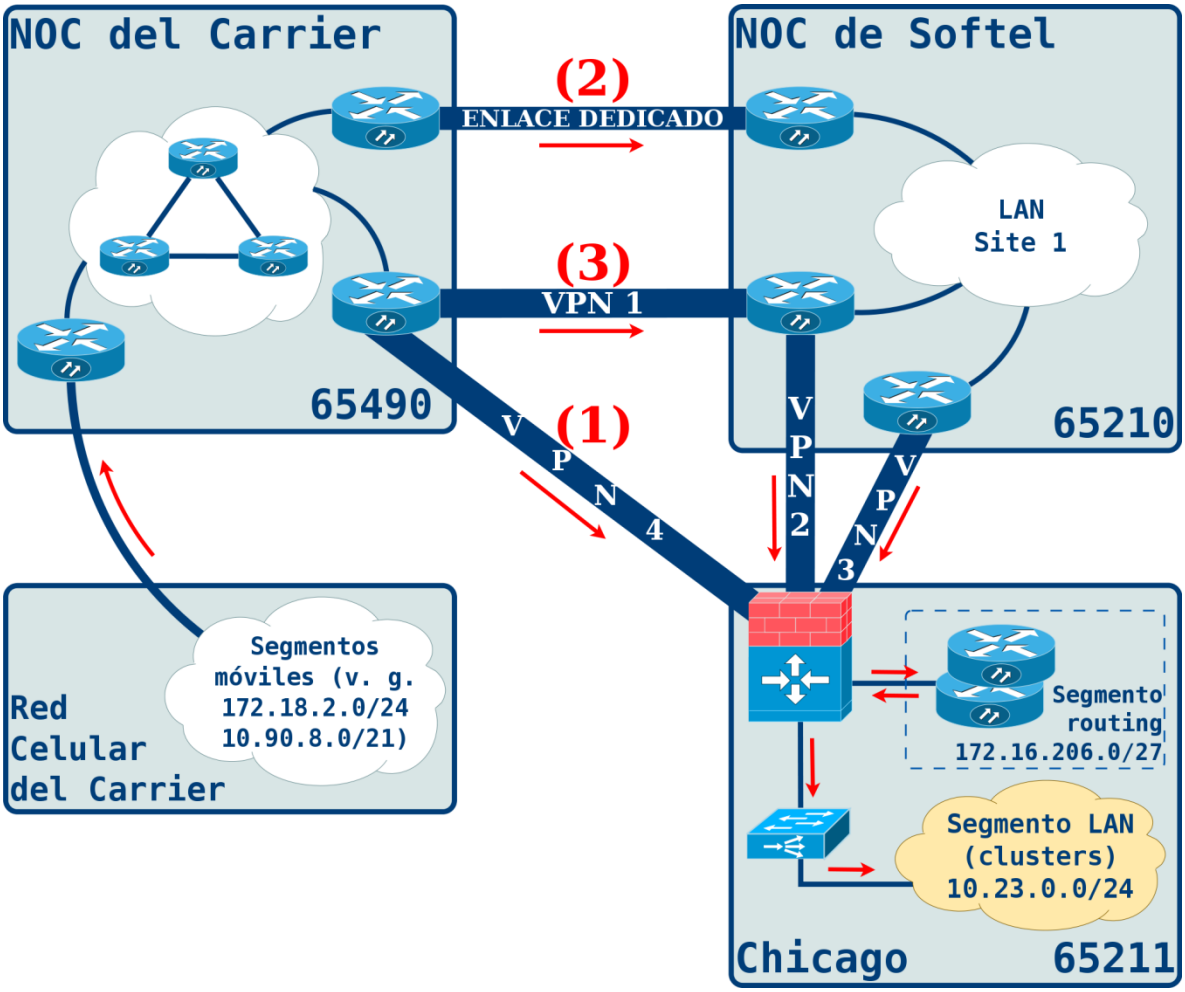

Figura 4.11. Posibles trayectorias
desde el *carrier* hacia el *site* 2.

Desde el *site* 1 hacia el *carrier*, los usuarios pueden alcanzar los segmentos IP de los SIM a través de cualquiera de los dos medios (enlace dedicado o VPN) dependiendo de cuál sea el *gateway* para el nodo interesado en comunicarse, ya que cada *router* frontera tiene a su propio medio como prioritario, en parte debido a los criterios de selección de rutas en BGP expuestos en el capítulo 2. Adicionalmente, en cada *router* se configuró un peso adecuado a sus respectivos vecinos para asegurar dicha situación.

Como una tercera y cuarta opciones, en caso de que los dos enlaces principales fallen, se tiene la posibilidad de alcanzar a los SIM a través de Chicago mediante la VPN2 y, en última instancia, por la VPN3. Para lograrlo se manipuló el valor de *local preference*, *weight* y, en algunos casos, se agregaron intencionalmente más saltos al *path* de las rutas propagadas por ciertos vecinos. La razón por la cual se eligieron esas prioridades fue una decisión interna en la compañía para tener un orden bien definido. Las prioridades son las mismas cuando se quiere llegar al segmento de LAN 10.23.0.0/24 que se encuentra en

Chicago: primero por la VPN2 luego por la VPN3. Vea la Figura 4.12.

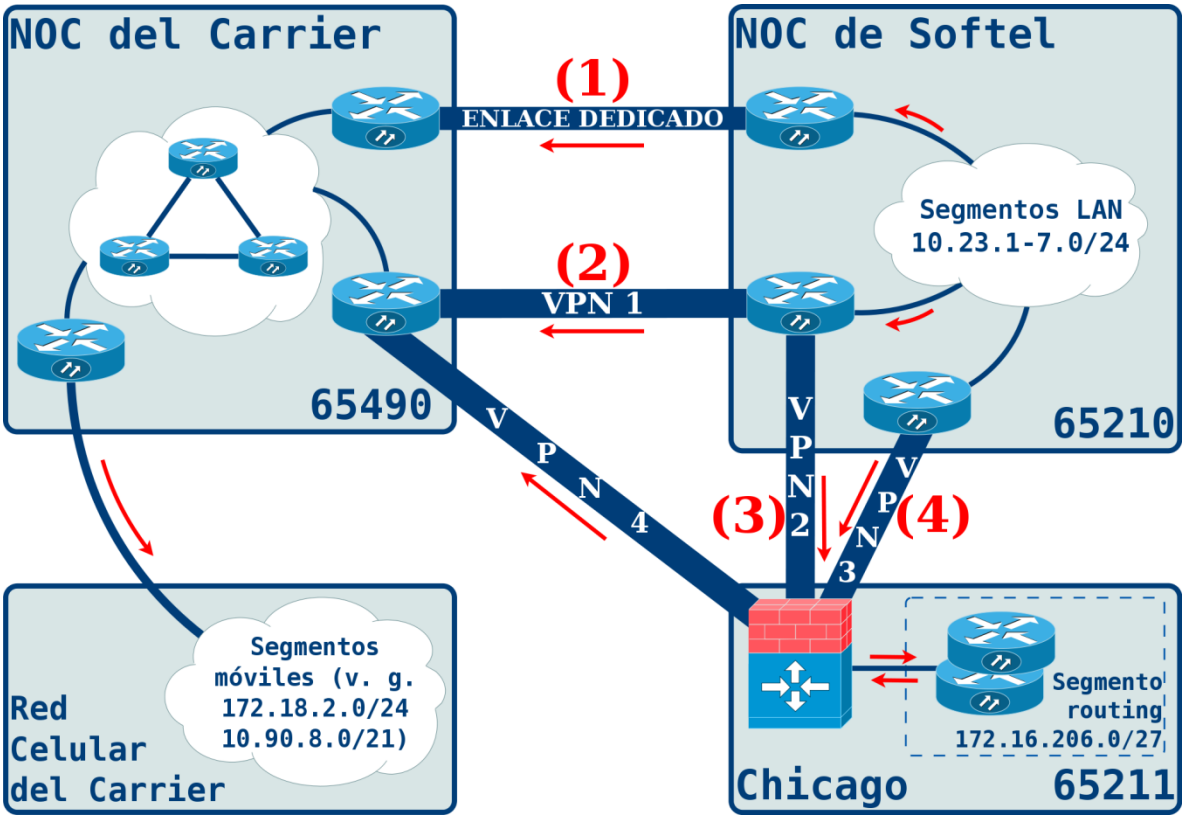

Figura 4.12. Posibles trayectorias desde el *site* 1 hacia el *carrier*.

Por último, en la Figura 4.13 se observa la definición de prioridades en la red de Chicago. Cuando la información viaja hacia el *carrier* se estableció el túnel de la VPN4 como principal; y, si éste falla, la preferencia es recíproca con respecto al *site* 1 tanto para alcanzar los dispositivos móviles como los segmentos de LAN presentes en el NOC de La Empresa.

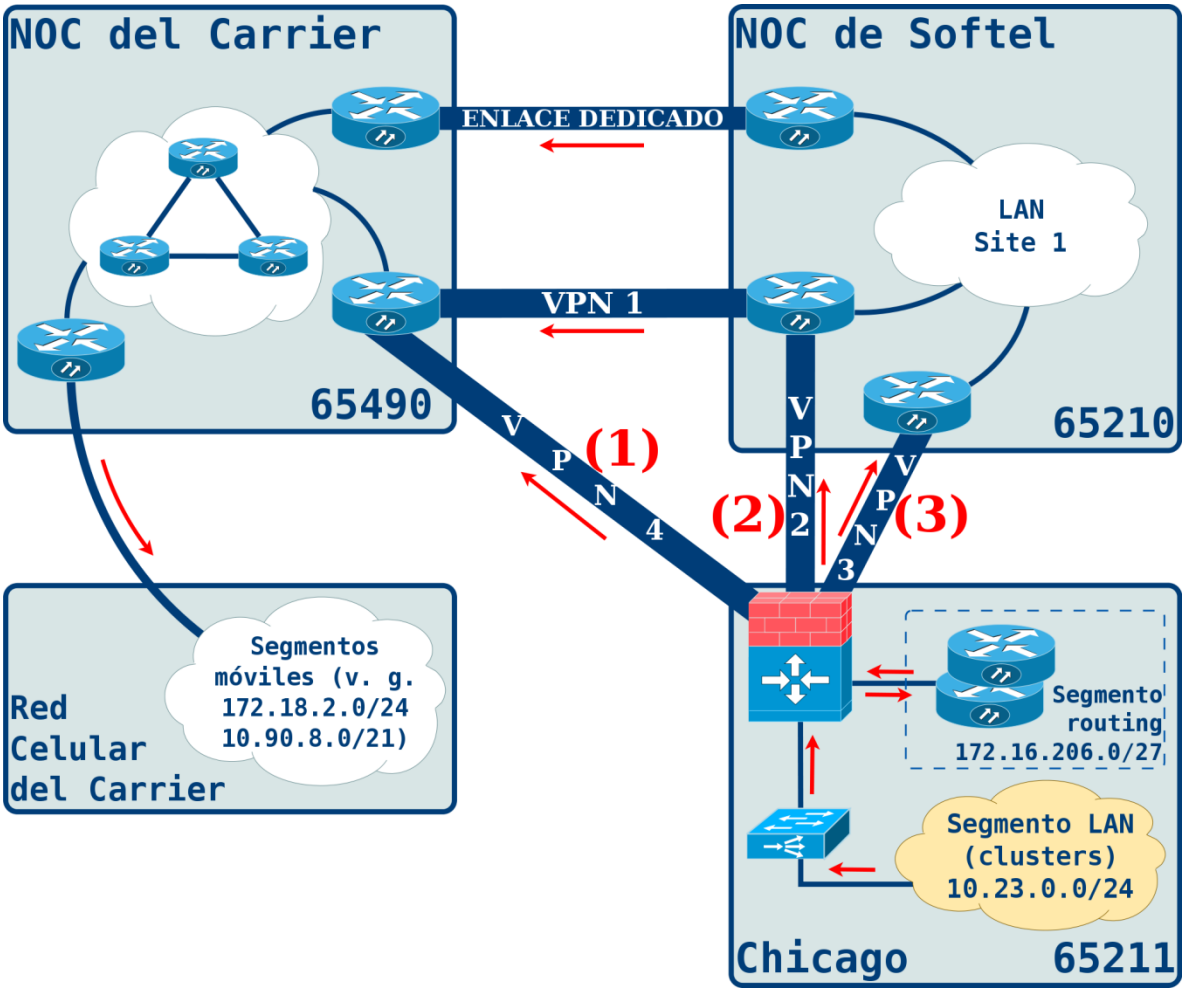

Figura 4.13. Posibles trayectorias desde el *site* 2 hacia el *carrier*.

Se debe tomar en cuenta que el *hold time* es equivalente a 3 períodos de *keepalive*, de 60 segundos cada uno. Considerando lo anterior, se puede intuir que la duración máxima de un evento de falla de comunicación será igual a 3 minutos, después de los cuales el proceso de BGP recalculará las rutas en base a la nueva topología y actualizará la tabla de *routing* en cada dispositivo involucrado.

La recuperación de un enlace caído no debe significar ningún problema de intermitencia ni la comunicación debe verse comprometida por ello. La vuelta a "*online*" de un medio que estaba fuera de línea implica solamente que la sesión BGP entre los elementos vecinos se establezca y que las rutas se compartan nuevamente entre ellos para proceder a la actualización de las RIBs en todos los demás elementos de la red a los que afecte el cambio. Dicha actualización es transparente para los usuarios y las comunicaciones en general. Si el enlace que había sido afectado es el principal, el fenómeno que ocurre es que de un segundo a otro los paquetes son enviados por una

interfaz diferente, sin modificar la constitución de los mismos. Si el enlace en falla era un secundario, no habrá cambio alguno puesto que todo va por el primario antes y después de que el evento ocurra.

4.4.2 *Clusters*.

Para manejar toda la lógica y la gestión del *cluster* se utilizó Pacemaker y Corosync, cuyas bondades se explicaron también en el capítulo 2. La combinación de las dos herramientas posibilita el "diálogo" entre los nodos pertenecientes al *cluster* y el control de los recursos administrados por el mismo.

Pacemaker cuenta con una interfaz de comandos en donde es posible modificar toda la configuración del *cluster* desde una terminal del Sistema Operativo. La ventaja es que la mayor parte de los comandos ingresados en la terminal de texto son automáticamente replicados al resto de los nodos participantes, salvo excepciones muy particulares.

La arquitectura construida a partir de los comandos en terminal es almacenada en un archivo de configuración que guarda toda la información en formato XML. El archivo se actualiza instantáneamente en función de los cambios que el administrador del *cluster* haga.

Pacemaker basa su funcionamiento en la existencia de "*resource agents*", que son archivos de código y comandos ejecutables para administrar los recursos de un *cluster*. Realmente no existe una definición específica del concepto de recurso para un *cluster*, más allá de: "lo que sea que un *cluster* sea capaz de administrar, es un recurso". Los recursos de un *cluster* pueden incluir direcciones IP, sistemas de archivos, servicios de base de datos y máquinas virtuales enteras, por ejemplificar sólo algunos.

Un *resource agent* puede ser escrito en cualquier lenguaje de programación, aunque la mayoría de ellos están hechos como *shell scripts* y en este proyecto no hubo una excepción.

La constitución de los *resource agents* puede tener diversas características que permiten tener un control integral sobre los servicios que prestan los nodos. Capacidades tales como la preferencia de un nodo sobre otro para ejecutar un programa, la colocación forzada de un recurso en el mismo nodo que alberga a otro, el orden en el que se arrancan o se detienen los servicios −entre otras−, son posibles gracias a que el *script* del *resource agent* debe tener, como mínimo, la capacidad de arrancar, detener y comprobar el estado de un recurso, de tal suerte que el *cluster* hace una revisión periódica y determina si el recurso en el nodo sigue activo y si además funciona correctamente. Cuando Pacemaker detecta problemas en el recurso de un *cluster* activo-pasivo, detiene el servicio en el nodo afectado e inicia el proceso de migración para arrancarlo en algún otro nodo disponible.

Ya aclarado lo anterior, en los siguientes puntos se puede explicar el comportamiento que tienen los *clusters* de La Empresa.

4.4.2.1 *Cluster* web.

El demonio más importante en este caso es el de Apache2, por lo que se debe dar de alta un recurso en el *cluster* para controlarlo. Éste no es el único recurso disponible, se debe complementar con otros que permitan tener un ambiente más completo. Puntualmente Apache2 es un recurso que está ligado a dos más en nuestro *cluster*: un recurso de "*ping*" y otro que proporciona una dirección IP virtual.

El recurso *ping* ayuda a comprobar que el sistema puede comunicarse con los demás elementos de la red; al menos hasta el nivel de la capa de internet del modelo TCP/IP. La operación del *ping* consiste en enviar un conjunto de paquetes ICMP de tipo *echo request* hacia algún domino o dirección IP y verificar que es alcanzable al recibir una respuesta satisfactoria (ICMP *echo reply*). Para el *cluster* web se decidió enviar los paquetes al *default gateway* de la DMZ.

La creación de una dirección IP virtual como recurso es muy parecido al trabajo que realiza VRRP, pero no se utilizó dicha herramienta porque es importante relacionar directamente la dirección IP virtual con la ejecución del demonio de Apache2 en el mismo servidor.

En adelante, todo es configuración mediante la edición de archivos o ejecutando comandos en una terminal de texto. El orden de arranque de los recursos debe obedecer a la siguiente lógica:

1) El recurso *ping* se pone en marcha en el nodo preferente y verifica que tenga comunicación con su *default gateway*.
2) Ya comprobada la comunicación, el nodo asume la dirección IP virtual del *cluster*.
3) Pacemaker verifica el *status* del proceso de Apache2 en ambos nodos y lo detiene, si es que estuviera activo, para reiniciarlo únicamente en el nodo principal.
4) Pacemaker verifica que el recurso de Apache2 esté activo y se encuentre funcionando correctamente en el nodo de interés.
5) El proceso de verificación tanto de Apache2, como de *ping* y de la dirección IP virtual se realizará periódicamente una vez que el *cluster* esté estable.

En caso de que alguno de los pasos anteriores falle, se detendrán todos los recursos en el nodo principal y se realizará el mismo proceso en el nodo secundario, con la ayuda de Corosync.

4.4.2.2 *Cluster* de base de datos.

En este *cluster* los elementos involucrados y su funcionamiento son prácticamente idénticos a los de web, salvo que ahora aparece PostgreSQL como recurso principal y un recurso adicional de sistema de archivos, en el cual se comparte una ruta del *filesystem* donde se monta el volumen lógico de almacenamiento administrado por LVM.

La construcción del *cluster* de base de datos y su operación normal se describe en los siguientes pasos:

1) Creé la arquitectura de LVM adecuada para administrar los elementos físicos y lógicos de almacenamiento.
2) Modifiqué la configuración necesaria para que DRBD funcione y sincronice la información entre los volúmenes de almacenamiento.
3) Configuré el servicio de PostgreSQL para que almacene y sincronice la información en la nueva estructura creada con LVM.
4) Asigné un recurso de dirección IP virtual asociada al nodo principal del *cluster*, para que reciba y procese todas las conexiones a las bases de datos.
5) En condiciones iniciales del *cluster*, Pacemaker revisa el *status* del proceso de PostgreSQL en todos los nodos y lo detiene, si resulta estar activo, para arrancarlo solamente en el nodo principal, quien ya tiene la dirección IP virtual asignada.
6) Pacemaker comprueba que el recurso de PostgreSQL ahora esté activo y se encuentre funcionando correctamente.
7) El proceso de verificación de PostgreSQL, del sistema de archivos compartido y de la dirección IP virtual se hará periódicamente una vez que el *cluster* se encuentre estable. Un error en cualquiera de estos servicios provocará la migración en cadena de todos ellos al nodo secundario.

4.5 Instalación y configuración.

Para poner en marcha todo el sistema descrito en las secciones pasadas, fue necesario apoyarse en un buen número de herramientas de *software*, sobre todo en el Sistema Operativo Debian 6. El IOS de Cisco ya cuenta con todo lo necesario para desempeñar el trabajo requerido, salvo por alguna licencia de seguridad que debe instalarse y que no se cubrirá en este informe, ya que el proveedor facilita las instrucciones al momento de adquirir el equipo.

Los paquetes que deben ser instalados en los *routers* y en los servidores con Linux se mencionan a partir de la próxima sección. Adicionalmente pueden incluirse algunos utensilios informáticos que permiten un mejor diagnóstico de parte del administrador y

facilitan el control del sistema. Herramientas como son: *sniffers*, agentes de SNMP (*Simple Network Management Protocol*), visualizadores de procesos del sistema, etcétera.

El *software* en un Sistema Operativo Debian se instala con la herramienta para gestión de paquetes "apt-get". Para mayores detalles sobre su uso, es recomendable consultar el siguiente *how-to* elaborado por la comunidad de Ubuntu: https://help.ubuntu.com/community/AptGet/Howto.

La función y uso de cada paquete mencionado se puede encontrar fácilmente en el manual que viene integrado en el sistema operativo para todos los comandos importantes (comando "man"), o bien, en la misma documentación de Debian: https://www.debian.org/distrib/packages.

Los apéndices A y B incluyen los archivos de configuración más importantes que utilicé en los routers Linux y las rutas donde se ubican dentro de su sistema de archivos.

En el caso de los Cisco ISR de ambos sites, los apéndices C y D contienen la sección de la configuración que introduje con la CLI del IOS.

Por último, los apéndices E y F muestran los comandos que deben ejecutarse en la terminal de Pacemaker para construir los *clusters*, los archivos de configuración básicos para los mismos y los *resource agents* de Apache2 y PostgreSQL que programé específicamente para el proyecto.

4.5.1 *Software* para un *router* Linux.

Los paquetes esenciales para que los *routers* con Debian 6 funcionen de acuerdo a la solución diseñada, deben ser los siguientes:

- linux-headers-$(uname -r)
- openswan
- openswan-modules-dkms
- quagga
- vrrpd

Herramientas adicionales se pueden obtener con la instalación de los siguientes paquetes:

- bridge-utils
- htop
- iftop
- curl
- ntpdate

- snmp
- snmpd
- tcpdump
- traceroute

4.5.2 *Software* para un servidor web Linux.

Un servidor web para *cluster* con Sistema Operativo Debian 6 debe tener los siguientes paquetes elementales para su puesta en marcha:

- apache2
- corosync
- curl
- linux-headers-$(uname -r)
- pacemaker

Herramientas adicionales se pueden obtener con la instalación de los siguientes paquetes:

- htop
- iftop
- ntpdate
- snmp
- snmpd
- tcpdump
- traceroute

4.5.3 *Software* para un servidor de base de datos Linux.

Para que un servidor de base de datos funcione en *cluster* con Sistema Operativo Debian 6 debe contar, al menos, con los siguientes paquetes:

- corosync
- linux-headers-$(uname -r)
- pacemaker
- postgresql

- postgresql-contrib
- xfsprogs

Los paquetes auxiliares pueden ser los mismos que se mencionaron para complementar un servidor web.

4.6 Comportamiento y resultados.

Se ha terminado de explicar la columna vertebral del proyecto que está siendo documentado en este informe, comenzando por una base teórica lo suficientemente amplia para no dejar de lado los conceptos importantes, seguida por un plan de desarrollo para seleccionar los instrumentos apropiados durante la implementación y proponiendo con ello un diseño más minucioso en donde se aterrizaron los detalles que definen el comportamiento y la operación del producto final.

Toca el turno de exponer los resultados obtenidos luego de realizar toda la configuración, coordinar las pruebas conjuntas en línea con el *carrier*, revisando que todos los escenarios de falla contemplados en el diseño están cubiertos y comprobando que el sistema continúa dando servicio con normalidad para los clientes, a pesar de las desfavorables circunstancias.

La sección de resultados está dividida en dos partes: la primera contempla información que arrojan todos los *routers* que participan en el esquema de redundancia WAN, de parte de La Empresa, ocultando una parte de las direcciones IP involucradas por razones de seguridad y privacidad, pero evidenciando que el diseño y los parámetros obtenidos coinciden; en la segunda parte se imprime la información de consulta de *status* que arrojan los *clusters* web y de base de datos.

Se muestran sólo las imágenes bajo un escenario normal, sin fallas, y se explican los datos importantes que ahí aparecen. Las impresiones de pantalla donde se comprueba que el sistema continúa en operación a pesar de la existencia de problemas en diferentes puntos de la red, no se incluyen debido a las gestiones administrativas que implican ese tipo de pruebas, las cuales se realizan con ventanas de mantenimiento rigurosamente controladas y con la previa autorización de todos los involucrados.

4.6.1 Redundancia WAN.

En este punto se incluyen impresiones de pantalla sobre la información de BGP, tablas de *routing*, túneles GRE y túneles IPsec, con excepción de la VPN en el *site* de Chicago, ya que esa no es administrada por La Empresa.

- RouterC1.

Este *router* es el encargado de recibir y enviar información directamente por el enlace dedicado en el *site* 1, por lo tanto no cuenta con un túnel IPsec ni GRE; únicamente la sesión de BGP.

Información de BGP.

Se puede ver en la Figura 4.14 la información y estadísticas de la vecindad con el *router* ubicado al otro lado del enlace dedicado *Frame Relay*. Entre muchas otras cosas, se aprecia la dirección IP del vecino, el AS al que pertenece y el número de prefijos (segmentos) alcanzables a través de él.

```
BGP neighbor is 192.168.252.25,  remote AS 65490, external link
  BGP version 4, remote router ID 192.168.[redacted]
  BGP state = Established, up for 11w3d
  Last read 00:00:08, last write 00:00:29, hold time is 180, keepalive interval is 60 seconds
  Neighbor sessions:
    1 active, is not multisession capable (disabled)
  Neighbor capabilities:
    Route refresh: advertised and received(new)
    Four-octets ASN Capability: advertised and received
    Address family IPv4 Unicast: advertised and received
    Graceful Restart Capability: received
      Remote Restart timer is 120 seconds
      Address families advertised by peer:
        IPv4 Unicast (was not preserved
    Enhanced Refresh Capability: advertised and received
    Multisession Capability:
    Stateful switchover support enabled: NO for session 1
  Message statistics:
    InQ depth is 0
    OutQ depth is 0

                         Sent       Rcvd
    Opens:                  1          1
    Notifications:          0          0
    Updates:             3440         11
    Keepalives:        125265     126965
    Route Refresh:          0          0
    Total:             128706     126977
  Default minimum time between advertisement runs is 30 seconds

 For address family: IPv4 Unicast
  Session: 192.168.252.25
  BGP table version 13951, neighbor version 13951/0
  Output queue size : 0
  Index 10, Advertise bit 1
  10 update-group member
  Inbound path policy configured
  Outbound path policy configured
  Route map for incoming advertisements is telcelin
  Route map for outgoing advertisements is telcelout
  Default weight 100
  Slow-peer detection is disabled
  Slow-peer split-update-group dynamic is disabled
                               Sent       Rcvd
  Prefix activity:             ----       ----
    Prefixes Current:            11         12 (Consumes 768 bytes)
    Prefixes Total:            1743         30
    Implicit Withdraw:           15         18
    Explicit Withdraw:         1717          0
    Used as bestpath:           n/a         12
    Used as multipath:          n/a          0
```

Figura 4.14. Información de vecinos eBGP en RouterC1.

En la tabla de la Figura 4.15 se muestra la información más específica de las rutas aprendidas gracias al *router* vecino. La columna "Network" indica la subred anunciada y en la columna "Next Hop" aparece la dirección IP del *router* al que deben reenviarse los paquetes con destino a esa subred. Note que existen tantas direcciones de "siguiente salto" como trayectorias posibles para encaminar los paquetes. En la parte izquierda se señala la ruta preferida por el sistema (después de hacer todo el proceso de selección de acuerdo a las reglas de BGP) con el símbolo ">".

Por último, la columna "Path" ofrece el mapeo de los Sistemas Autónomos por los que un paquete tiene que pasar para llegar a la subred aprendida.

```
BGP table version is 13823, local router ID is 10.  .1.
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale, m multipath, b backup-path, f RT-Filter,
              x best-external, a additional-path, c RIB-compressed,
Origin codes: i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found

     Network          Next Hop            Metric LocPrf Weight Path
 * i 10.  .0/23      10.  .1.                         100      0 65490 64900 64900 64542 i
 *>                   192.168.                                100 65490 64900 64542 i
 * i 10.  .0/24      10.  .1.                         100      0 65490 64900 64900 64542 i
 *>                   192.168.                                100 65490 64900 64542 i
 * i 10.  .0/20      10.  .1.                         100      0 65490 64900 64900 64542 i
 *>                   192.168.                                100 65490 64900 64542 i
 * i 10.  .0/22      10.  .1.                         100      0 65490 64900 64900 64540 i
 *>                   192.168.                                100 65490 64900 64540 i
 * i 10.  .0/23      10.  .1.                         100      0 65490 64900 64900 64540 i
 *>                   192.168.                                100 65490 64900 64540 i
 * i 10.  .0/24      10.  .1.                         100      0 65490 64900 64900 64540 i
 *>                   192.168.                                100 65490 64900 64540 i
 * i 10.  .0/21      10.  .1.                         100      0 65490 64900 64900 64542 i
 *>                   192.168.                                100 65490 64900 64542 i
 * i 10.  .0/21      10.  .1.                         100      0 65490 64900 64900 64542 64542 64542 64542 i
 *>                   192.168.                                100 65490 64900 64542 64542 64542 64542 i
 * i 1  .0/23        10.  .1.                         100      0 65490 64900 64900 64542 i
 *>                   192.168.                                100 65490 64900 64542 i
 * i 1  .0/23        10.  .1.                         100      0 65490 64900 64900 64542 i
 *>                   192.168.                                100 65490 64900 64542 i
 * i 1  .0/21        10.  .1.                         100      0 65490 64900 64900 64542 i
 *>                   192.168.                                100 65490 64900 64542 i
 * i 1  .0/21        10.  .1.                         100      0 65490 64900 64900 64542 i
 *>                   192.168.                                100 65490 64900 64542 i
```

Figura 4.15. Tabla de prefijos recibidos por BGP en RouterC1.

Los prefijos compartidos desde La Empresa al *carrier* se observan en la Figura 4.16.

```
BGP table version is 13823, local router ID is 10.[redacted].1.[redacted]
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale, m multipath, b backup-path, f RT-Filter,
              x best-external, a additional-path, c RIB-compressed,
Origin codes: i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found

     Network          Next Hop            Metric LocPrf Weight Path
 *>i 10.[redacted].0.0/24     10.[redacted].1.[redacted]          0    100      0 65211 ?
 * i                   10.[redacted].1.[redacted]          0    100      0 65211 ?
 * i 10.[redacted].1.0/24     10.[redacted].1.[redacted]          0    100      0 i
 * i                   10.[redacted].1.[redacted]          0    100      0 i
 * i                   10.[redacted].1.[redacted]          0    100      0 i
 * i                   10.[redacted].1.[redacted]          0    100      0 i
 * i                   10.[redacted].1.[redacted]          0    100      0 i
 *>                    0.0.0.0                  0         32768 i
 *>i 10.[redacted].2.0/24     10.[redacted].1.[redacted]          0    100      0 ?
 * i                   10.[redacted].1.[redacted]          1    100      0 ?
 *>i 10.[redacted].3.0/24     10.[redacted].1.[redacted]          0    100      0 ?
 * i                   10.[redacted].1.[redacted]          1    100      0 ?
 *>i 10.[redacted].4.0/24     10.[redacted].1.[redacted]          1    100      0 ?
 *>i 10.[redacted].5.0/24     10.[redacted].1.[redacted]          1    100      0 ?
```

Figura 4.16. Tabla de prefijos anunciados por BGP en RouterC1.

- RouterI5.

Este es el *router* con la configuración más extensa y con mayor información para consultar. En él se construyen las VPN 1 y 2 con sus respectivos túneles GRE: uno hacia el *carrier* de telefonía celular y otro hacia Chicago.

Información de IPsec.

De acuerdo a la topología utilizada, lo primero que debe funcionar correctamente en este *router* son los túneles IPsec. Las capturas en las Figuras 4.17 y 4.18 indican que las asociaciones de fase 1 y 2 están establecidas con el *carrier* de telefonía celular y con el *site* de Chicago, respectivamente.

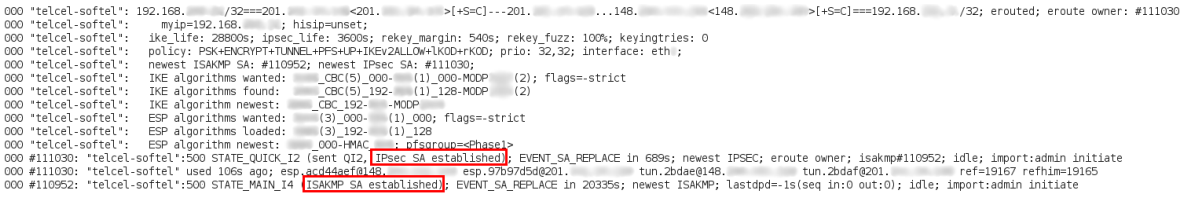

Figura 4.17. *Status* de la VPN con el *carrier* en RouterI5.

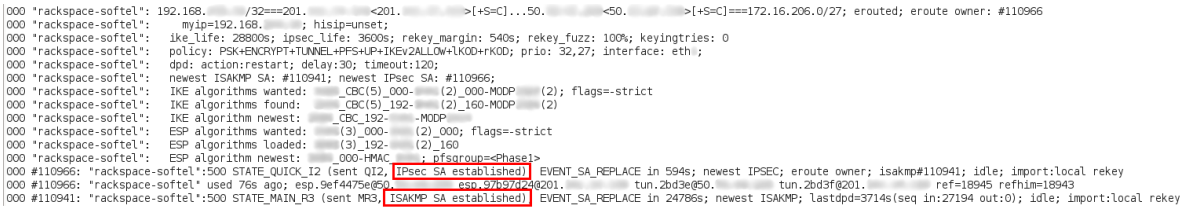

Figura 4.18. *Status* de la VPN con el *site* 2 en RouterI5.

Información de túneles GRE.

Posterior a la revisión de las VPN, se debe verificar que los túneles GRE estén construidos y que los extremos sean los dominios de cifrado de las VPN. Ver Figura 4.19.

```
tunRackspace: gre/ip  remote 172.16.206.13  local 192.168.[illegible]  ttl 255
tun10: gre/ip  remote 192.168.[illegible]  local 192.168.[illegible]  ttl 255
```

Figura 4.19. Definición de los túneles GRE en RouterI5.

En las Figuras 4.20 y 4.21 se ve a detalle la información de las interfaces virtuales para cada uno de los túneles, destacando su dirección IP. Para el túnel hacia el *site* 2 se eligió la subred 192.168.205.0/30, cuyo valor del último octeto en este *router* es 1 y en el de Chicago es 2.

```
tunRackspace Link encap:UNSPEC  HWaddr C0-A8-FA-21-55-08-00-00-00-00-00-00-00-00-00-00
          inet addr:192.168.205.1  P-t-P:192.168.205.1  Mask:255.255.255.252
          UP POINTOPOINT RUNNING NOARP  MTU:16236  Metric:1
          RX packets:14123910 errors:0 dropped:0 overruns:0 frame:0
          TX packets:844299 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:885527877 (844.5 MiB)  TX bytes:58656864 (55.9 MiB)
```

Figura 4.20. Información de la interfaz del túnel hacia el *site* 2 en RouterI5.

La dirección IP del túnel que va hacia el *carrier* en este router es 192.168.41.6/30, mientras que la interfaz del router en el NOC del *carrier* tiene la dirección 192.168.41.5/30

```
tun10     Link encap:UNSPEC  HWaddr C0-A8-FA-16-48-08-00-00-00-00-00-00-00-00-00-00
          inet addr:192.168.41.6  P-t-P:192.168.41.6  Mask:255.255.255.252
          UP POINTOPOINT RUNNING NOARP  MTU:1400  Metric:1
          RX packets:1483127 errors:0 dropped:0 overruns:0 frame:0
          TX packets:328378713 errors:1 dropped:0 overruns:0 carrier:1
          collisions:0 txqueuelen:0
          RX bytes:104646364 (99.7 MiB)  TX bytes:4043408256 (3.7 GiB)
```

Figura 4.21. Información de la interfaz del túnel hacia el *carrier* en RouterI5.

Información de BGP.

A continuación se incluyen las descripciones de los vecinos BGP, comenzando por el de Chicago en la Figura 4.22 y posteriormente el del *carrier*, con la Figura 4.23. Identificarlos es sencillo gracias a su dirección IP y número de AS.

```
BGP neighbor is 192.168.205.2, remote AS 65211, local AS 65210  external link
  BGP version 4, remote router ID 172.16.206.
  BGP state = Established, up for 6d16h32m
  Last read 01:20:42, hold time is 180, keepalive interval is 60 seconds
  Neighbor capabilities:
    4 Byte AS: advertised and received
    Route refresh: advertised and received(old & new)
    Address family IPv4 Unicast: advertised and received
  Message statistics:
    Inq depth is 0
    Outq depth is 0
                         Sent       Rcvd
    Opens:                  6          4
    Notifications:          1          4
    Updates:             3106         44
    Keepalives:        103703     114192
    Route Refresh:          0          0
    Capability:             0          0
    Total:             106816     114244
  Minimum time between advertisement runs is 30 seconds
  Default weight 100

 For address family: IPv4 Unicast
  Community attribute sent to this neighbor(both)
  Outbound path policy configured
  Route map for outgoing advertisements is *rackspace
  14 accepted prefixes

  Connections established 6; dropped 5
  Last reset 6d16h32m, due to BGP Notification received
Local host: 192.168.205.1, Local port: 52747
Foreign host: 192.168.205.2, Foreign port: 179
Nexthop: 192.168.205.1
Nexthop global: fe80::202:b6ff:fe42:536
Nexthop local: ::
BGP connection: non shared network
Read thread: on  Write thread: off
```

Figura 4.22. Información del vecino eBGP del *site* 2 en RouterI5.

```
BGP neighbor is 192.168.41.5, remote AS 65490, local AS 65210, external link
  BGP version 4, remote router ID 192.168.[redacted]
  BGP state = Established, up for 10w2d01h
  Last read 01:20:55, hold time is 180, keepalive interval is 60 seconds
  Neighbor capabilities:
    4 Byte AS: advertised
    Route refresh: advertised and received(old & new)
    Address family IPv4 Unicast: advertised and received
  Message statistics:
    Inq depth is 0
    Outq depth is 0
                         Sent       Rcvd
    Opens:                  1          1
    Notifications:          0          0
    Updates:             3072         10
    Keepalives:        103770     103780
    Route Refresh:          0          0
    Capability:             0          0
    Total:             106843     103791
  Minimum time between advertisement runs is 30 seconds
  Default weight 100

 For address family: IPv4 Unicast
  Community attribute sent to this neighbor(both)
  Inbound path policy configured
  Outbound path policy configured
  Route map for incoming advertisements is *telcelin
  Route map for outgoing advertisements is *telcelout
  12 accepted prefixes

  Connections established 1; dropped 0
  Last reset never
Local host: 192.168.41.6, Local port: 37530
Foreign host: 192.168.41.5, Foreign port: 179
Nexthop: 192.168.41.6
Nexthop global: fe80::202:b6ff:fe42:536
Nexthop local: ::
BGP connection: non shared network
Read thread: on  Write thread: off
```

Figura 4.23. Información del vecino eBGP del *carrier* en RouterI5.

Note, a partir de la siguiente tabla de BGP (Figura 4.24), que este *router* recibe diferentes opciones para alcanzar los segmentos de las tarjetas SIM. La ruta anunciada por el vecino con dirección IP 192.168.41.5, que pertenece al *carrier*, es la que intencionalmente tiene más peso. Como segunda opción (inferida por el número de saltos de AS) aparece el *router* del enlace dedicado y el tercer lugar lo tiene el *Next Hop* con dirección IP 192.168.205.2, situado en Chicago.

```
BGP table version is 0, local router ID is 10.[illegible].1.[illegible]
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale, R Removed
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
*  10.[illegible].0/23   192.168.205.2                      100 65211 65490 64900 64900 64542 i
*>                   192.168.41.5                        100 65490 64900 64900 64542 i
* i                   10.[illegible].1.[illegible]              0    100      0 65490 64900 64542 i
*  10.[illegible].0/24   192.168.205.2                      100 65211 65490 64900 64900 64542 i
*>                   192.168.41.5                        100 65490 64900 64900 64542 i
* i                   10.[illegible].1.[illegible]              0    100      0 65490 64900 64542 i
*  10.[illegible].0/20   192.168.205.2                      100 65211 65490 64900 64900 64542 i
*>                   192.168.41.5                        100 65490 64900 64900 64542 i
* i                   10.[illegible].1.[illegible]              0    100      0 65490 64900 64542 i
*  10.[illegible].0/22   192.168.205.2                      100 65211 65490 64900 64900 64540 i
*>                   192.168.41.5                        100 65490 64900 64900 64540 i
* i                   10.[illegible].1.[illegible]              0    100      0 65490 64900 64540 i
*  10.[illegible].0/23   192.168.205.2                      100 65211 65490 64900 64900 64540 i
*>                   192.168.41.5                        100 65490 64900 64900 64540 i
* i                   10.[illegible].1.[illegible]              0    100      0 65490 64900 64540 i
*  10.[illegible].0/24   192.168.205.2                      100 65211 65490 64900 64900 64540 i
*>                   192.168.41.5                        100 65490 64900 64900 64540 i
* i                   10.[illegible].1.[illegible]              0    100      0 65490 64900 64540 i
*  10.[illegible].0/21   192.168.205.2                      100 65211 65490 64900 64900 64542 i
*>                   192.168.41.5                        100 65490 64900 64900 64542 i
* i                   10.[illegible].1.[illegible]              0    100      0 65490 64900 64542 i
*  10.[illegible].0/21   192.168.205.2                      100 65211 65490 64900 64900 64542 64542 64542 64542 i
*>                   192.168.41.5                        100 65490 64900 64900 64542 64542 64542 64542 i
* i                   10.[illegible].1.[illegible]              0    100      0 65490 64900 64542 64542 64542 64542 i
*  148.[illegible].0/23  192.168.205.2                      100 65211 65490 64900 64900 64542 i
*>                   192.168.41.5                        100 65490 64900 64900 64542 i
* i                   10.[illegible].1.[illegible]              0    100      0 65490 64900 64542 i
*  148.[illegible].0/23  192.168.205.2                      100 65211 65490 64900 64900 64542 i
*>                   192.168.41.5                        100 65490 64900 64900 64542 i
* i                   10.[illegible].1.[illegible]              0    100      0 65490 64900 64542 i
*  148.[illegible].0/21  192.168.205.2                      100 65211 65490 64900 64900 64542 i
*>                   192.168.41.5                        100 65490 64900 64900 64542 i
* i                   10.[illegible].1.[illegible]              0    100      0 65490 64900 64542 i
*  148.[illegible].0/21  192.168.205.2                      100 65211 65490 64900 64900 64542 i
*>                   192.168.41.5                        100 65490 64900 64900 64542 i
* i                   10.[illegible].1.[illegible]              0    100      0 65490 64900 64542 i
Total number of prefixes 12
```

Figura 4.24. Tabla de prefijos recibidos por BGP en RouterI5.

La tabla mostrada en la Figura 4.25 lista los prefijos compartidos por BGP que pertenecen a la LAN de La Empresa. El segmento 10.23.0.0/24 es alcanzable directamente por el *router* de Chicago (192.168.205.2) y también a través del *router* vecino que cuenta con la segunda VPN hacia el *site* 2.

```
BGP table version is 0, local router ID is 10.[redacted].1.[redacted]
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale, R Removed
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
* i10.[redacted].0.0/24     10.[redacted].1.[redacted]              0    100      0 65211 ?
*>                  192.168.205.2            0             100 65211 ?
* i10.[redacted].1.0/24     10.[redacted].1.[redacted]              0    100      0 i
* i                  10.[redacted].1.[redacted]              0    100      0 i
* i                  10.[redacted].1.[redacted]              0    100      0 i
* i                  10.[redacted].1.[redacted]              0    100      0 i
* i                  10.[redacted].1.[redacted]              0    100      0 i
*                   0.0.0.0                  1         32768 ?
*>                  0.0.0.0                  0         32768 i
* i10.[redacted].2.0/24     10.[redacted].1.[redacted]              1    100      0 ?
*>                  10.[redacted].1.[redacted]              0         32768 ?
* i10.[redacted].3.0/24     10.[redacted].1.[redacted]              1    100      0 ?
*>                  10.[redacted].1.[redacted]              0         32768 ?
*>i10.[redacted].4.0/24     10.[redacted].1.[redacted]              1    100      0 ?
*> 10.[redacted].5.0/24     0.0.0.0                  1         32768 ?
```

Figura 4.25. Tabla de prefijos anunciados por BGP en RouterI5.

Se puede consultar muy fácilmente en la tabla de *routing* cuál es el camino preferente hacia las diferentes subredes que se conocen por BGP. Observe la Figura 4.26: la segunda ruta indica que a la subred 10.23.0.0/24 se llega mediante la dirección 192.168.205.2 y ésta, a su vez, pertenece a la subred administrativa del túnel hacia Chicago, como lo indica la primera ruta.

```
192.168.205.0/30 dev tunRackspace  proto kernel  scope link  src 192.168.205.1
10.[redacted].0.0/24 via 192.168.205.2 dev tunRackspace  proto zebra
```

Figura 4.26. Rutas hacia el *site* 2 por el túnel GRE en RouterI5.

La misma consulta se hizo en la Figura 4.27 para los segmentos móviles, conocidos a través de la dirección administrativa del túnel en el extremo del *carrier* (primera ruta).

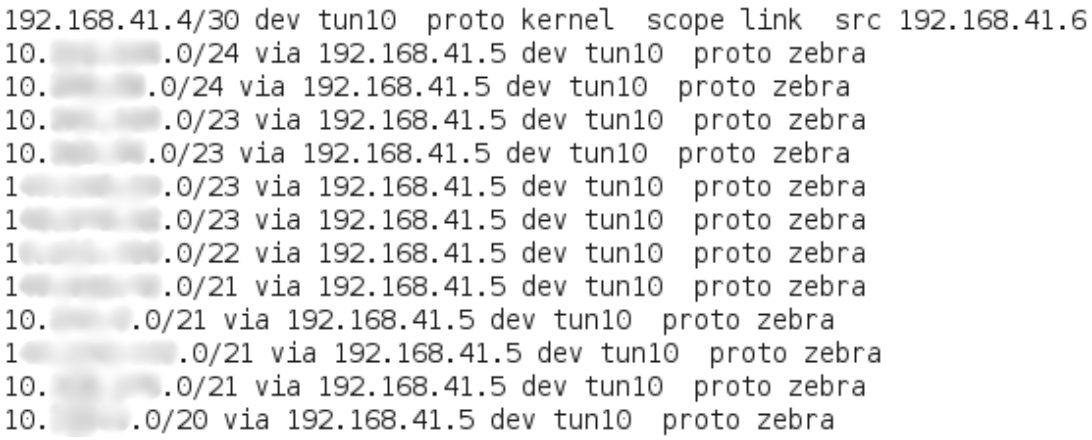

Figura 4.27. Rutas hacia el *carrier* por
el túnel GRE en RouterI5.

- RouterI4.

Este *router* es utilizado para comunicarse con Chicago. En él sólo hay un túnel IPsec importante dentro de este proyecto (VPN3), en el cual reside un túnel GRE y una sesión BGP.

Información de IPsec.

Al igual que con el RouterI5, se debe verificar el estado del túnel IPsec. La Figura 4.28 indica que tanto la fase 1 como la fase 2 están establecidas correctamente.

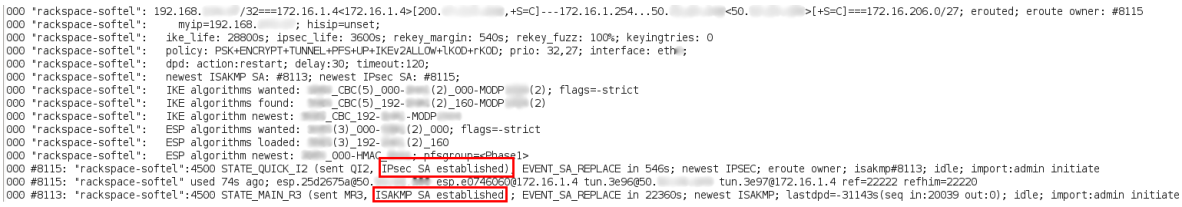

Figura 4.28. *Status* de la VPN con el *carrier* en RouterI4.

Una comprobación adicional puede ser la ruta que se tiene para alcanzar el "segmento *routing*" (172.16.206.0/27) de Chicago a través de la interfaz "ipsec0", como aparece en la Figura 4.29.

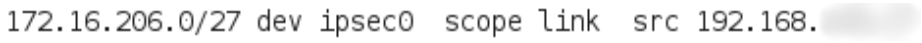

Figura 4.29. Ruta de IPsec generada por
la VPN hacia el *site* 2 en RouterI4.

Información de túneles GRE.

El túnel GRE tiene como *peers* los dominios de cifrado de la VPN. Ver la Figura 4.30.

```
tunRackspace: gre/ip  remote 172.16.206.13  local 192.168.[illegible]  ttl 255
```

Figura 4.30. Definición del túnel GRE en RouterI4.

En la Figura 4.31, la consulta de información en la interfaz del túnel imprime su dirección IP, situada dentro de la red 192.168.205.4/30. El último octeto para este *router* tiene un valor decimal de 5; por otro lado, la misma interfaz en el *router* de Chicago tiene la dirección 192.168.205.6.

```
tunRackspace Link encap:UNSPEC  HWaddr C0-A8-FA-25-3F-09-00-00-00-00-00-00-00-00-00-00
          inet addr:192.168.205.5  P-t-P:192.168.205.5  Mask:255.255.255.252
          UP POINTOPOINT RUNNING NOARP  MTU:16236  Metric:1
          RX packets:13430931 errors:0 dropped:0 overruns:0 frame:0
          TX packets:28408447 errors:7 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:568046558 (541.7 MiB)  TX bytes:1894689218 (1.7 GiB)
```

Figura 4.31. Información de la interfaz del túnel hacia el *site* 2 en RouterI4.

Información de BGP.

Los datos recabados del vecino BGP en Chicago muestran, en la Figura 4.32, su dirección IP, los AS remoto y local y cuántos prefijos han sido aceptados.

```
BGP neighbor is 192.168.205.6, remote AS 65211, local AS 65210, external link
  BGP version 4, remote router ID 172.16.206.
  BGP state = Established, up for 2d18h36m
  Last read 06:36:55, hold time is 180, keepalive interval is 60 seconds
  Neighbor capabilities:
    4 Byte AS: advertised and received
    Route refresh: advertised and received(old & new)
    Address family IPv4 Unicast: advertised and received
  Message statistics:
    Inq depth is 0
    Outq depth is 0
                         Sent       Rcvd
    Opens:                  1          1
    Notifications:          0          0
    Updates:              108          6
    Keepalives:          3998       4397
    Route Refresh:          0          0
    Capability:             0          0
    Total:               4107       4404
  Minimum time between advertisement runs is 30 seconds

 For address family: IPv4 Unicast
  Community attribute sent to this neighbor(both)
  Outbound path policy configured
  Route map for outgoing advertisements is *rackspace
  14 accepted prefixes

  Connections established 1; dropped 0
  Last reset never
Local host: 192.168.205.5, Local port: 33956
Foreign host: 192.168.205.6, Foreign port: 179
Nexthop: 192.168.205.5
Nexthop global: fe80::202:b6ff:fe43:516d
Nexthop local: ::
BGP connection: non shared network
Read thread: on  Write thread: off
```

Figura 4.32. Información del vecino eBGP del *site* 2 en RouterI4.

La tabla BGP de prefijos compartidos, en la Figura 4.33, arroja las direcciones de las subredes donde se encuentran los SIM, así como los prefijos de LAN que se están anunciando a los AS exteriores.

```
BGP table version is 0, local router ID is 10.[redacted].1.[redacted]
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale, R Removed
Origin codes: i - IGP, e - EGP, ? - incomplete

   Network          Next Hop            Metric LocPrf Weight Path
* i10.[redacted].1.0/24     10.[redacted].1.[redacted]          0    100      0 i
* i                         10.[redacted].1.[redacted]          0    100      0 i
* i                         10.[redacted].1.[redacted]          0    100      0 i
* i                         10.[redacted].1.[redacted]          0    100      0 i
* i                         10.[redacted].1.[redacted]          0    100      0 i
*>                          0.0.0.0                             0         32768 i
*>i10.[redacted].2.0/24     10.[redacted].1.[redacted]          0    100      0 ?
* i                         10.[redacted].1.[redacted]          1    100      0 ?
*>i10.[redacted].3.0/24     10.[redacted].1.[redacted]          0    100      0 ?
* i                         10.[redacted].1.[redacted]          1    100      0 ?
*>i10.[redacted].4.0/24     10.[redacted].1.[redacted]          1    100      0 ?
*>i10.[redacted].5.0/24     10.[redacted].1.[redacted]          1    100      0 ?
* i10.[redacted].0/23       10.[redacted].1.[redacted]               100      0 65490 64900 64900 64542 i
*>i                         10.[redacted].1.[redacted]          0    100      0 65490 64900 64542 i
*                           192.168.205.6                                    0 65211 65490 64900 64900 64542 i
* i10.[redacted].0/24       10.[redacted].1.[redacted]               100      0 65490 64900 64900 64542 i
*>i                         10.[redacted].1.[redacted]          0    100      0 65490 64900 64542 i
*                           192.168.205.6                                    0 65211 65490 64900 64900 64542 i
* i10.[redacted].0/20       10.[redacted].1.[redacted]               100      0 65490 64900 64900 64542 i
*>i                         10.[redacted].1.[redacted]          0    100      0 65490 64900 64542 i
*                           192.168.205.6                                    0 65211 65490 64900 64900 64542 i
* i10.[redacted].0/22       10.[redacted].1.[redacted]               100      0 65490 64900 64900 64540 i
*>i                         10.[redacted].1.[redacted]          0    100      0 65490 64900 64540 i
*                           192.168.205.6                                    0 65211 65490 64900 64900 64540 i
* i10.[redacted].0/23       10.[redacted].1.[redacted]               100      0 65490 64900 64900 64540 i
*>i                         10.[redacted].1.[redacted]          0    100      0 65490 64900 64540 i
*                           192.168.205.6                                    0 65211 65490 64900 64900 64540 i
* i10.[redacted].0/24       10.[redacted].1.[redacted]               100      0 65490 64900 64900 64540 i
*>i                         10.[redacted].1.[redacted]          0    100      0 65490 64900 64540 i
*                           192.168.205.6                                    0 65211 65490 64900 64900 64540 i
* i10.[redacted].0/21       10.[redacted].1.[redacted]               100      0 65490 64900 64900 64542 i
*>i                         10.[redacted].1.[redacted]          0    100      0 65490 64900 64542 i
*                           192.168.205.6                                    0 65211 65490 64900 64900 64542 i
* i10.[redacted].0/21       10.[redacted].1.[redacted]               100      0 65490 64900 64900 64542 64542 64542 64542 i
*>i                         10.[redacted].1.[redacted]          0    100      0 65490 64900 64542 64542 64542 64542 i
*                           192.168.205.6                                    0 65211 65490 64900 64900 64542 64542 64542 64542 i
* i1[redacted].0/23         10.[redacted].1.[redacted]               100      0 65490 64900 64900 64542 i
*>i                         10.[redacted].1.[redacted]          0    100      0 65490 64900 64542 i
*                           192.168.205.6                                    0 65211 65490 64900 64900 64542 i
* i1[redacted].0/23         10.[redacted].1.[redacted]               100      0 65490 64900 64900 64542 i
*>i                         10.[redacted].1.[redacted]          0    100      0 65490 64900 64542 i
*                           192.168.205.6                                    0 65211 65490 64900 64900 64542 i
* i1[redacted].0/21         10.[redacted].1.[redacted]               100      0 65490 64900 64900 64542 i
*>i                         10.[redacted].1.[redacted]          0    100      0 65490 64900 64542 i
*                           192.168.205.6                                    0 65211 65490 64900 64900 64542 i
```

Figura 4.33. Tabla de prefijos recibidos y anunciados por BGP en RouterI4.

- RackSRouter1 y RackSRouter2.

Recordemos que en Chicago hay 2 Cisco ISR 1921 con una configuración prácticamente idéntica. El *router* activo tiene un túnel GRE hacia el NOC del *carrier* y dos más hacia los *routers* de La Empresa (RouterI4 y RouterI5) en la Ciudad de México.

Los datos del proceso de VRRP y su *status* se pueden leer en la Figura 4.34. Ahí se indica claramente cuál es la dirección IP virtual del *router*, la dirección MAC virtual, el grupo VRRP al que pertenece, su prioridad, la descripción del proceso, etcétera.

```
GigabitEthernet0/0 - Group 10
Virtual IP between RackSRouter1 and RackSRouter2.
  State is Master
  Virtual IP address is 172.16.206.13
  Virtual MAC address is 0000.5e00.010a
  Advertisement interval is 1.000 sec
  Preemption enabled
  Priority is 250
  Master Router is 172.16.206.11 (local), priority is 250
  Master Advertisement interval is 1.000 sec
  Master Down interval is 3.023 sec
```

Figura 4.34. Información de VRRP en RackSRouter1.

Información de túneles GRE.

Después de que el túnel IPsec termina en el *firewall* de nuestro proveedor, los Cisco ISR administrados por La Empresa reciben los paquetes provenientes del exterior ya sin cifrado, pero encapsulados en GRE. En total hay tres túneles GRE dedicados para esta solución y cuya descripción a detalle se muestra enseguida.

La interfaz Tunnel0 pertenece al GRE de la VPN2, de manera que el *peer* de este túnel se localiza en el RouterI5 de la Ciudad de México y sus direcciones administrativas están dentro de la red 192.168.205.0/30, como ya se había descrito. Véase Figura 4.35.

```
Tunnel0 is up, line protocol is up
  Hardware is Tunnel
  Description: GRE Tunnel to Softel Router VPN 1.
  Internet address is 192.168.205.2/30
  MTU 17916 bytes, BW 100 Kbit/sec, DLY 50000 usec,
     reliability 255/255, txload 7/255, rxload 1/255
  Encapsulation TUNNEL, loopback not set
  Keepalive not set
  Tunnel source 172.16.206.13, destination 192.168.
  Tunnel protocol/transport GRE/IP
    Key disabled, sequencing disabled
    Checksumming of packets disabled
  Tunnel TTL 255, Fast tunneling enabled
  Tunnel transport MTU 1476 bytes
  Tunnel transmit bandwidth 8000 (kbps)
  Tunnel receive bandwidth 8000 (kbps)
  Last input 00:00:28, output 00:00:28, output hang never
  Last clearing of "show interface" counters 49w4d
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
  Output queue: 0/0 (size/max)
  5 minute input rate 0 bits/sec, 0 packets/sec
  5 minute output rate 3000 bits/sec, 7 packets/sec
     5389797 packets input, 767374943 bytes, 0 no buffer
     Received 0 broadcasts (0 IP multicasts)
     0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
     22131969 packets output, 2962432908 bytes, 0 underruns
     0 output errors, 0 collisions, 0 interface resets
     0 unknown protocol drops
     0 output buffer failures, 0 output buffers swapped out
```

Figura 4.35. Información de la interfaz del túnel hacia RouterI5 en RackSRouter1.

El túnel establecido hacia el RouterI4 se identifica con la interfaz Tunnel1 y su dirección administrativa es 192.168.205.6/30, tal como aparece en la Figura 4.36.

```
Tunnel1 is up, line protocol is up
  Hardware is Tunnel
  Description: GRE Tunnel to Softel Router VPN 2.
  Internet address is 192.168.205.6/30
  MTU 17916 bytes, BW 100 Kbit/sec, DLY 50000 usec,
     reliability 255/255, txload 1/255, rxload 7/255
  Encapsulation TUNNEL, loopback not set
  Keepalive not set
  Tunnel source 172.16.206.13, destination 192.168.
  Tunnel protocol/transport GRE/IP
    Key disabled, sequencing disabled
    Checksumming of packets disabled
  Tunnel TTL 255, Fast tunneling enabled
  Tunnel transport MTU 1476 bytes
  Tunnel transmit bandwidth 8000 (kbps)
  Tunnel receive bandwidth 8000 (kbps)
  Last input 00:00:00, output 00:00:00, output hang never
  Last clearing of "show interface" counters 49w4d
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
  Output queue: 0/0 (size/max)
  5 minute input rate 3000 bits/sec, 6 packets/sec
  5 minute output rate 0 bits/sec, 0 packets/sec
     34085292 packets input, 2521799859 bytes, 0 no buffer
     Received 0 broadcasts (0 IP multicasts)
     0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
     14590062 packets output, 1465032471 bytes, 0 underruns
     0 output errors, 0 collisions, 0 interface resets
     0 unknown protocol drops
     0 output buffer failures, 0 output buffers swapped out
```

Figura 4.36. Información de la interfaz del túnel hacia RouterI4 en RackSRouter1.

Por último, en la Figura 4.37 está la interfaz Tunnel11, que va hacia el *carrier*. La subred de administración a la que pertenece este túnel es 192.168.41.8/30, siendo la segunda dirección de *host* (.10) para nuestro *router*.

```
Tunnel11 is up, line protocol is up
  Hardware is Tunnel
  Description: GRE Tunnel to Telcel.
  Internet address is 192.168.41.10/30
  MTU 17916 bytes, BW 100 Kbit/sec, DLY 50000 usec,
     reliability 255/255, txload 10/255, rxload 10/255
  Encapsulation TUNNEL, loopback not set
  Keepalive not set
  Tunnel source 172.16.206.13, destination 192.168.
  Tunnel protocol/transport GRE/IP
    Key disabled, sequencing disabled
    Checksumming of packets disabled
  Tunnel TTL 255, Fast tunneling enabled
  Tunnel transport MTU 1476 bytes
  Tunnel transmit bandwidth 8000 (kbps)
  Tunnel receive bandwidth 8000 (kbps)
  Last input 00:00:02, output 00:00:11, output hang never
  Last clearing of "show interface" counters 48w5d
  Input queue: 0/75/0/0 (size/max/drops/flushes); Total output drops: 0
  Queueing strategy: fifo
  Output queue: 0/0 (size/max)
  5 minute input rate 4000 bits/sec, 10 packets/sec
  5 minute output rate 4000 bits/sec, 9 packets/sec
     47664692 packets input, 3508137739 bytes, 0 no buffer
     Received 0 broadcasts (0 IP multicasts)
     0 runts, 0 giants, 0 throttles
     0 input errors, 0 CRC, 0 frame, 0 overrun, 0 ignored, 0 abort
     44866516 packets output, 3282335758 bytes, 0 underruns
     0 output errors, 0 collisions, 0 interface resets
     0 unknown protocol drops
     0 output buffer failures, 0 output buffers swapped out
```

Figura 4.37. Información de la interfaz del túnel hacia el *carrier* en RackSRouter1.

Información de BGP.

Las capturas de las Figuras 4.38, 4.39 y 4.40 muestran la información de los vecinos eBGP, fácilmente identificables por sus direcciones IP y el AS al que pertenecen. Destaca el número de prefijos recibidos y anunciados, además de la cantidad de memoria RAM que éstos consumen en el *router*.

```
BGP neighbor is 192.168.205.1,  remote AS 65210, external link
  BGP version 4, remote router ID 10.  .1.
  BGP state = Established, up for 1w2d
  Last read 00:00:51, last write 00:00:47, hold time is 180, keepalive interval is 60 seconds
  Neighbor sessions:
    1 active, is not multisession capable (disabled)
  Neighbor capabilities:
    Route refresh: advertised and received(new)
    Four-octets ASN Capability: advertised and received
    Address family IPv4 Unicast: advertised and received
    Enhanced Refresh Capability: advertised
    Multisession Capability:
    Stateful switchover support enabled: NO for session 1
  Message statistics:
    InQ depth is 0
    OutQ depth is 0

                         Sent       Rcvd
    Opens:                  1          1
    Notifications:          0          0
    Updates:                6        551
    Keepalives:         14892      13531
    Route Refresh:          0          0
    Total:              14899      14083
  Default minimum time between advertisement runs is 30 seconds

 For address family: IPv4 Unicast
  Session: 192.168.205.1
  BGP table version 15949, neighbor version 15949/0
  Output queue size : 0
  Index 101, Advertise bit 2
  101 update-group member
  Outbound path policy configured
  Route map for outgoing advertisements is softeluno
  Slow-peer detection is disabled
  Slow-peer split-update-group dynamic is disabled
                               Sent       Rcvd
  Prefix activity:             ----       ----
    Prefixes Current:            14         29 (Consumes 1856 bytes)
    Prefixes Total:              14        301
    Implicit Withdraw:            0          0
    Explicit Withdraw:            0        272
    Used as bestpath:           n/a         16
    Used as multipath:          n/a          0
```

Figura 4.38. Información eBGP de RouterI5 en RackSRouter1.

```
BGP neighbor is 192.168.205.5,  remote AS 65210, external link
  BGP version 4, remote router ID 10.  .1.
  BGP state = Established, up for 2d18h
  Last read 00:00:18, last write 00:00:11, hold time is 180, keepalive interval is 60 seconds
  Neighbor sessions:
    1 active, is not multisession capable (disabled)
  Neighbor capabilities:
    Route refresh: advertised and received(new)
    Four-octets ASN Capability: advertised and received
    Address family IPv4 Unicast: advertised and received
    Enhanced Refresh Capability: advertised
    Multisession Capability:
    Stateful switchover support enabled: NO for session 1
  Message statistics:
    InQ depth is 0
    OutQ depth is 0

                         Sent       Rcvd
    Opens:                  1          1
    Notifications:          0          0
    Updates:                6        109
    Keepalives:          4415       4013
    Route Refresh:          0          0
    Total:               4422       4123
  Default minimum time between advertisement runs is 30 seconds

 For address family: IPv4 Unicast
  Session: 192.168.205.5
  BGP table version 15949, neighbor version 15949/0
  Output queue size : 0
  Index 103, Advertise bit 1
  103 update-group member
  Outbound path policy configured
  Route map for outgoing advertisements is softeldos
  Slow-peer detection is disabled
  Slow-peer split-update-group dynamic is disabled
                               Sent       Rcvd
  Prefix activity:             ----       ----
    Prefixes Current:            14         29 (Consumes 1856 bytes)
    Prefixes Total:              14         79
    Implicit Withdraw:            0          0
    Explicit Withdraw:            0         50
    Used as bestpath:           n/a          1
    Used as multipath:          n/a          0
```

Figura 4.39. Información eBGP de RouterI4 en RackSRouter1.

```
BGP neighbor is 192.168.41.9,  remote AS 65490, external link
  BGP version 4, remote router ID 192.168.
  BGP state = Established, up for 5w1d
  Last read 00:00:51, last write 00:00:36, hold time is 180, keepalive interval is 60 seconds
  Neighbor sessions:
    1 active, is not multisession capable (disabled)
  Neighbor capabilities:
    Route refresh: advertised and received(new)
    Four-octets ASN Capability: advertised
    Address family IPv4 Unicast: advertised and received
    Enhanced Refresh Capability: advertised
    Multisession Capability:
    Stateful switchover support enabled: NO for session 1
  Message statistics:
    InQ depth is 0
    OutQ depth is 0

                         Sent       Rcvd
    Opens:                  1          1
    Notifications:          0          0
    Updates:             2479          6
    Keepalives:         56757      52719
    Route Refresh:          0          0
    Total:              59237      52726
  Default minimum time between advertisement runs is 30 seconds

 For address family: IPv4 Unicast
  Session: 192.168.41.9
  BGP table version 15949, neighbor version 15949/0
  Output queue size : 0
  Index 96, Advertise bit 3
  96 update-group member
  Inbound path policy configured
  Outbound path policy configured
  Route map for incoming advertisements is telcelin
  Route map for outgoing advertisements is telcelout
  Default weight 100
  Slow-peer detection is disabled
  Slow-peer split-update-group dynamic is disabled
                               Sent       Rcvd
  Prefix activity:             ----       ----
    Prefixes Current:             8         12 (Consumes 768 bytes)
    Prefixes Total:            1776         18
    Implicit Withdraw:         1039          6
    Explicit Withdraw:          729          0
    Used as bestpath:           n/a         12
    Used as multipath:          n/a          0
```

Figura 4.40. Información del vecino eBGP del *carrier* en RackSRouter1.

Se puede notar, gracias a la tabla de BGP en la Figura 4.41, que las subredes de los SIM tienen como ruta principal la anunciada por el *router* del *carrier* con dirección 192.168.41.9. Las otras dos opciones son a través de los túneles del *site* 1. La dirección 192.168.205.1 tiene un peso mayor, por ende es la secundaria y las trayectorias conocidas por del vecino 192.168.205.5 son las terciarias.

```
BGP table version is 15949, local router ID is 172.16.206.
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale, m multipath, b backup-path, f RT-Filter,
              x best-external, a additional-path, c RIB-compressed,
Origin codes: i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found

     Network          Next Hop            Metric LocPrf Weight Path
 *   10.[illegible].0/23   192.168.205.5                      0 65210 65490 64900 64542 i
 *                     192.168.205.1                         50 65210 65490 64900 64900 64542 i
 *>                    192.168.41.9                         100 65490 64900 64900 64542 i
 *   10.[illegible].0/24   192.168.205.5                      0 65210 65490 64900 64542 i
 *                     192.168.205.1                         50 65210 65490 64900 64900 64542 i
 *>                    192.168.41.9                         100 65490 64900 64900 64542 i
 *   10.[illegible].0/20   192.168.205.5                      0 65210 65490 64900 64542 i
 *                     192.168.205.1                         50 65210 65490 64900 64900 64542 i
 *>                    192.168.41.9                         100 65490 64900 64900 64542 i
 *   10.[illegible].0/22   192.168.205.5                      0 65210 65490 64900 64540 i
 *                     192.168.205.1                         50 65210 65490 64900 64900 64540 i
 *>                    192.168.41.9                         100 65490 64900 64900 64540 i
 *   10.[illegible].0/23   192.168.205.5                      0 65210 65490 64900 64540 i
 *                     192.168.205.1                         50 65210 65490 64900 64900 64540 i
 *>                    192.168.41.9                         100 65490 64900 64900 64540 i
 *   10.[illegible].0/24   192.168.205.5                      0 65210 65490 64900 64540 i
 *                     192.168.205.1                         50 65210 65490 64900 64900 64540 i
 *>                    192.168.41.9                         100 65490 64900 64900 64540 i
 *   10.[illegible].0/21   192.168.205.5                      0 65210 65490 64900 64542 i
 *                     192.168.205.1                         50 65210 65490 64900 64900 64542 i
 *>                    192.168.41.9                         100 65490 64900 64900 64542 i
 *   10.[illegible].0/21   192.168.205.5                      0 65210 65490 64900 64542 64542 64542 64542 i
 *                     192.168.205.1                         50 65210 65490 64900 64900 64542 64542 64542 64542 i
 *>                    192.168.41.9                         100 65490 64900 64900 64542 64542 64542 64542 i
 *   1[illegible].0/23   192.168.205.5                      0 65210 65490 64900 64542 i
 *                     192.168.205.1                         50 65210 65490 64900 64900 64542 i
 *>                    192.168.41.9                         100 65490 64900 64900 64542 i
 *   1[illegible].0/23   192.168.205.5                      0 65210 65490 64900 64542 i
 *                     192.168.205.1                         50 65210 65490 64900 64900 64542 i
 *>                    192.168.41.9                         100 65490 64900 64900 64542 i
 *   1[illegible].0/21   192.168.205.5                      0 65210 65490 64900 64542 i
 *                     192.168.205.1                         50 65210 65490 64900 64900 64542 i
 *>                    192.168.41.9                         100 65490 64900 64900 64542 i
 *   1[illegible].0/21   192.168.205.5                      0 65210 65490 64900 64542 i
 *                     192.168.205.1                         50 65210 65490 64900 64900 64542 i
 *>                    192.168.41.9                         100 65490 64900 64900 64542 i
```

Figura 4.41. Tabla de prefijos de los móviles recibidos por BGP en RackSRouter1.

Cuando se trata de llegar a los segmentos de LAN que se anuncian desde el *site* en México, naturalmente existen dos opciones. Es importante notar en la Figura 4.42 que se agregó un salto adicional sobre su mismo AS en el *router* con dirección IP 192.168.205.5 (VPN3) para que sea el camino alternativo y la trayectoria por Tunnel0 sea la principal.

```
BGP table version is 15949, local router ID is 172.16.206.
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
              r RIB-failure, S Stale, m multipath, b backup-path, f RT-Filter,
              x best-external, a additional-path, c RIB-compressed,
Origin codes: i - IGP, e - EGP, ? - incomplete
RPKI validation codes: V valid, I invalid, N Not found

     Network          Next Hop            Metric LocPrf Weight Path
*>   10.  .0.0/24     172.16.206.1             0         32768 ?
*    10.  .1.0/24     192.168.205.5            0             0 65210 65210 i
*>                    192.168.205.1            0             0 65210 i
*    10.  .2.0/24     192.168.205.5                          0 65210 65210 ?
*>                    192.168.205.1            0             0 65210 ?
*    10.  .3.0/24     192.168.205.5                          0 65210 65210 ?
*>                    192.168.205.1            0             0 65210 ?
*    10.  .4.0/24     192.168.205.5                          0 65210 65210 ?
*>                    192.168.205.1                          0 65210 ?
*    10.  .5.0/24     192.168.205.5                          0 65210 65210 ?
*>                    192.168.205.1            1             0 65210 ?
```

Figura 4.42. Tabla de prefijos LAN recibidos por BGP en RackSRouter1.

4.6.2 *Clusters*.

Mediante comandos ejecutados en la terminal es posible obtener el *status* de los *clusters* en cualquier momento. Incluso también se pueden manipular directamente los recursos que administra el *cluster* para realizar tareas diversas, que pueden arrancar, detener, habilitar, deshabilitar, reiniciar, migrar, "congelar" y "descongelar" los recursos. Dichas operaciones se ejecutan con un comando que afecta a un recurso a la vez; no obstante, es importante conocer perfectamente al *cluster* para saber qué implicaciones tendrá una alteración en el estado de un recurso y cómo afectará al comportamiento del resto.

4.6.2.1 *Cluster* web.

De acuerdo a la Figura 4.43, al consultar el estado actual del *cluster* de servidores web en el nodo 524742-web2 se puede ver que ambos integrantes están "*Online*" y el nodo 524740-web1 es quien tiene activo el servicio "apache-svc".

```
Cluster Status for 1777225-clu2 @ Mon Jan 19 12:16:49 2015
Member Status: Quorate

 Member Name                 ID   Status
 ------ ----                 ---- ------
 524740-web1                     1 Online, rgmanager
 524742-web2                     2 Online, Local, rgmanage

 Service Name                 Owner (Last)                          State
 ------- ----                 ----- ------                          -----
 service:apache-svc           524740-web1                           started
```

Figura 4.43. Status del *cluster* web.

4.6.2.2 Cluster de base de datos.

Con relación al *cluster* de base de datos, en el resultado del status consultado en el nodo 524746-db2 se ve que los dos servidores están "*Online*" y es precisamente este mismo nodo quien tiene activos los servicios de base de datos (pgsql-svc) y de sistema de archivos (hanfs-svc).

```
Cluster Status for 1777225-clu1 @ Mon Jan 19 12:13:00 2015
Member Status: Quorate

 Member Name                 ID   Status
 ------ ----                 ---- ------
 524743-db1                      1 Online, rgmanager
 524746-db2                      2 Online, Local, rgmanager

 Service Name                 Owner (Last)                          State
 ------- ----                 ----- ------                          -----
 service:hanfs-svc            524746-db2                            started
 service:pgsql-svc            524746-db2                            started
```

Figura 4.44. Status del *cluster* de base de datos.

Capítulo 5.

Posibles mejoras.

Al igual que en cualquier otro sistema, existen diversos puntos de mejora que no se consideraron en el diseño inicial, ya sea por omisión, conveniencia o cualquier otra razón posible.

Tal como se enfatizó durante la planeación y el diseño, la idea fue conformar una solución que sea bastante flexible y, con la ayuda de nuevos conceptos y la inclusión de herramientas extra, gozar de nuevas capacidades que sirvan para explotar al máximo las propiedades del sistema.

Actualmente, ya con el sistema productivo, se está trabajando en la planeación de ciertos cambios que agreguen nuevas características en el ambiente de redundancia WAN y que al mismo tiempo contribuyan a mejorar las capacidades de los *clusters*. Los siguientes párrafos están dedicados a listar algunas de ellas y los avances que se tienen al respecto.

5.1 Próximos cambios para redundancia WAN.

Se ha hecho bastante énfasis en la cantidad de enlaces disponibles para enviar y recibir tráfico, así como la forma en la que todos se respaldan ante posibles caídas. Sin embargo, hasta ahora la conmutación por falla está basada en la jerarquía que intencionalmente se le da a los enlaces mediante la modificación de parámetros de BGP, tales como *weight*, *local preference* o *as-path prepend*. El objetivo ahora es encontrar la manera de hacer balanceo de carga entre los *routers* frontera de la compañía. BGP *Multipath* es una opción muy viable y utilizada para ese fin, por lo que se investigará más a fondo y con las pruebas de laboratorio pertinentes.

De antemano sabemos que del lado del *carrier* no será posible lograrlo por las condiciones que ellos mismos establecieron, pero desde La Empresa sería un buen avance controlar y distribuir la ocupación del ancho de banda en el enlace dedicado y las VPN. Se están buscando los parámetros y ajustes de BGP que lo permitan, además de evaluar la posibilidad de implementar un protocolo IGP más adecuado que facilite el trabajo. Tal podría ser el caso de OSPF, del cual se conoce su criterio y desempeño superiores a iBGP.

La distribución de los segmentos de LAN en La Empresa podría considerarse “rígida”, con subredes que sólo existen en un *site* o en otro. Se planea la inclusión de EoIP como un poderoso aliado para extender la red local de La Empresa y con ello fusionar ambos *sites*, de tal manera que segmentos idénticos existan de ambos sitios y puedan comunicarse a nivel de capa 2 OSI entre ellos. Para hacerlo posible, EoIP toma ventaja de GRE, del cual ya se comentaron ampliamente sus bondades para retransmitir tramas de *broadcast*, entre

otras propias de una LAN, convirtiendo una de las interfaces de red físicas de los *routers* en una interfaz virtual que actúa como un *bridge* de WAN.

La seguridad de las VPN, por otro lado, se encuentra hasta cierto punto limitada por el uso de frases pre-compartidas como modo de autenticación en fase 1, dada la posibilidad de vulnerar la red si éstas son conocidas por personas no autorizadas. Los cánones promueven el uso de certificados X.509 como un método mucho más seguro de identificación entre los *peers* de una VPN.

A continuación se presenta una breve explicación sobre el uso de certificados digitales y sus ventajas:

Los certificados digitales permiten que una entidad demuestre que es quien dice ser, es decir, que está en posesión de la clave secreta asociada a su certificado. En otras palabras, proporcionan un mecanismo para verificar la autenticidad de programas y documentos obtenidos a través de la red, el envío de mensajes cifrados o firmados digitalmente, el control de acceso a recursos, etcétera.

El formato de certificados X.509 es un estándar de la ITU-T (*International Telecommunication Union-Telecommunication Standarization Sector*) y la ISO/IEC (*International Standards Organization / International Electrotechnical Commission*), que especifica la posibilidad de controlar los periodos de validez de los certificados o anularlos si se ven comprometidos, además de conocer quién los genera y la infraestructura requerida para soportarlos.

Los certificados digitales sólo son útiles si existe alguna Autoridad Certificadora (CA: *Certification Authority*) que los valide. Si alguien se certifica a sí mismo no hay ninguna garantía de que su identidad sea la que anuncia y estrictamente no debería ser aceptada por otras organizaciones.

Es importante tener la capacidad de verificar que una Autoridad Certificadora ha emitido un certificado y detectar si un certificado no es válido. Para evitar la falsificación de certificados, la entidad certificadora firma el certificado digitalmente, después de autenticar la identidad de un sujeto mediante los procedimientos establecidos.

5.2 Futuras consideraciones para los *clusters*.

Ciertamente, la primera mejora que salta a la vista es para el *cluster* web, el cual debería ser de tipo activo-activo. De otra forma, se desperdicia mucha de la capacidad que puede tener el *cluster* como entidad de procesamiento.

El plan a futuro es que el *cluster* de servidores web desaparezca y en su lugar aprovechar las capacidades que tiene el balanceador de carga Brocade en las capas 3 y 4 OSI para tener un mejor rendimiento en la DMZ. El diseño para el cambio va más allá de los alcances de este reporte, pero se puede asegurar que para el desempeño que La Empresa

busca en la convivencia del servicio web con sus propios procesos, el balanceador es la opción más efectiva para potenciar el rendimiento y la disponibilidad del sistema.

Referente al *cluster* de base de datos, no hay demasiado qué hacer. El diseño es el adecuado y el rendimiento que tiene hasta ahora es bastante bueno. Como mejora se planea ampliarlo a un geo *cluster*, agregando al menos un nodo que lo respalde en la Ciudad de México. Para llevarlo a cabo, se ha pensado en la inclusión de un tercer sitio en México que sólo sirva como árbitro para satisfacer las características básicas que debe tener un *cluster* multi-*site*, explicadas en el capítulo 1. Aprovechando las ventajas de EoIP y los túneles de VPN que ya existen entre los *sites* 1 y 2, será posible el intercambio de mensajes entre los nodos del *cluster* usando Corosync. Los datos se van a almacenar también en un disco duro localizado en la Ciudad de México y su contenido debe ser una réplica exacta del LUN en Chicago. La sincronización entre los discos puede lograrse utilizando DRBD Proxy, un *software* para replicar bases de datos con alcances y características bastante interesantes, que a grandes rasgos se introdujeron en el capítulo 2.

Actualmente se sigue investigando más sobre el tema y participando en diferentes foros y listas de correo de LINBIT para entender y explotar el potencial de DRBD. A nivel conceptual está prácticamente listo, aunque la principal preocupación será el desempeño del sistema con latencias promedio superiores a las de una red Ethernet, tomando en cuenta que la distancia entre ambos sitios y la cantidad de *routers* y *switches* intermedios por los que transita la información es considerable. Las pruebas que se han realizado hasta ahora se hicieron en un ambiente local, con EoIP incluido en el ejercicio y consiguiendo resultados muy satisfactorios.

CONCLUSIONES.

La experiencia de participar activamente en un proyecto tan completo, que consistió en diseñar y construir un sistema de alta disponibilidad para aplicaciones de carácter crítico, me ha dejado un desarrollo importante y enriquecedor en diversos aspectos. Las áreas de crecimiento incluyen, desde el valioso aprendizaje teórico y práctico dentro de mi campo laboral, hasta la capacidad de relacionarme profesionalmente con distintas áreas de trabajo y otras organizaciones en donde las condiciones, los intereses involucrados y los riesgos que se corren son muy variados y afectan a un gran número de personas.

La labor necesaria para construir el sistema no fue poca ni fue sencilla; requirió de una buena cantidad de tiempo para comprenderlo, comenzando por adquirir una base sólida de conocimientos y reforzando también lo aprendido durante los cursos en la Facultad de Ingeniería de la UNAM.

A la par de la investigación se fue construyendo un modelo de desarrollo a lo largo de diversas reuniones entre los involucrados, que permitieron definir por completo el esquema final. Mediante ensayos de laboratorio, cursos, pruebas de escritorio y simulaciones se fue armando paso a paso cada una de las fases establecidas para el avance.

El orden, el empeño y la disciplina siempre serán indispensables para tener el mejor producto y la mejor calidad posibles, que se reflejan en los resultados alcanzados. Consecuentemente se logró el objetivo de tener un sistema confiable contra la mayor cantidad de circunstancias desfavorables.

Anteriormente se contaba ya con un diseño basado en *routing* estático, una redundancia menos fiable y la conmutación por falla no era totalmente automática; sin mencionar que no se tenía la flexibilidad necesaria para crecer la topología de una manera tan simple. Cualquier modificación, por mínima que fuere, implicaba reproducirla manualmente en cada *router* involucrado en el sistema, considerando sus condiciones muy particulares de acuerdo al papel que desempeñaba. El respaldo de toda la información de las bases de datos, antes de que existiera un *cluster*, se hacía en ventanas de mantenimiento programadas periódicamente y la conmutación de los servidores era manual al momento de presentarse un incidente. Esta situación daba como resultado un periodo de tiempo prácticamente inaceptable con el servicio y los portales fuera de línea, afectando severamente el porcentaje de disponibilidad.

Con el proyecto funcionando por completo, tenemos la tranquilidad de operar en un sistema mucho más robusto, que se comporta a la altura de las necesidades que un servicio de monitoreo para dispositivos M2M exige durante las 24 horas del día. La diferencia ha sido clara durante el tiempo que lleva funcionando; siendo conscientes de que las fallas de *hardware* y en la infraestructura de los servicios que contratamos no están exentas de suceder en cualquier momento. La operación de la redundancia y del *cluster* ha respondido como se esperaba y trabajaremos constantemente para mejorarla.

Está claro que el avance tecnológico y la creciente demanda de servicios en la rama de las Telecomunicaciones nos obligan cada vez más a mejorar e innovar en los sistemas que ya se tienen para hacerlos más fiables y eficientes, además de crear nuevas herramientas y adquirir dispositivos de mayor capacidad que hagan la vida de los usuarios aún más cómoda y, en consecuencia, tengamos consumidores satisfechos. Finalmente, puedo concluir que el objetivo de este proyecto ha sido cubierto cabalmente, con una metodología clara y exitosa que fue documentada con ayuda de este informe.

REFERENCIAS.

- ✓ *Clusters*:

Robert W. Lucke. “Building Clustered Linux Systems”

Karl Kopper. “The Linux Enterprise Cluster: Build a Highly Available Cluster with Commodity *Hardware* and Free *Software*”

Enrique Vargas, Joseph Bianco, David Deeths: "Sun Cluster Environment: Sun Cluster 2.2"

Charles Bookman. “Linux Clustering: Building and Maintaining Linux Clusters”

http://www.linux-ha.org/

https://www.suse.com/documentation/sle_ha/pdfdoc/book_sleha/book_sleha.pdf

- ✓ M2M:

David Boswarthick, Omar Elloumi, Olivier Hersent. "M2M Communications: A Systems Approach".

- ✓ Redes de datos:

STALLINGS, William. "Comunicaciones y redes de computadores", 6a. edición, Madrid. Prentice Hall, 2000.

TANENBAUM, Andrew. Wetherall, David. "Computer Networks", 5th edition. Prentice Hall

- ✓ Modelos de referencia:

TANENBAUM, Andrew. Wetherall, David. "Computer Networks", 5th edition. Prentice Hall

- ✓ Protocolo:

KUROSE, James. Keith, Ross. "Computer Networking: A Top-Down Approach. Featuring the Internet", 6th edition. Addison-Wesley, 2013

- ✓ Puerto:

http://es.wikipedia.org/wiki/Puerto_de_red

- ✓ Firewall:

TANENBAUM, Andrew. Wetherall, David. "Computer Networks", 5th edition. Prentice Hall

KUROSE, James. Keith, Ross. "Computer Networking: A Top-Down Approach. Featuring the Internet", 6th edition. Addison-Wesley, 2013

- ✓ DMZ:

TANENBAUM, Andrew. Wetherall, David. "Computer Networks", 5th edition. Prentice Hall

KUROSE, James. Keith, Ross. "Computer Networking: A Top-Down Approach. Featuring the Internet", 6th edition. Addison-Wesley, 2013

- ✓ VPN:

TANENBAUM, Andrew. Wetherall, David. "Computer Networks", 5th edition. Prentice Hall

http://www.bdat.net/documentos/cursos/ar01s612.html

- ✓ IPsec:

http://www.ipsec-howto.org/spanish/x161.html#IPSEC_MODE

- ✓ GRE:

http://tools.cisco.com/search/results/display?url=http%3a%2f%2fwww.cisco.com%2fc%2fen%2fus%2ftd%2fdocs%2frouters%2fasr9000%2fsoftware%2fasr9k_r5-1%2flxvpn%2fconfiguration%2fguide%2fvcasr9kv351%2fvcasr9k51gre.pdf&pos=3&query=generic+routing+encapsulation

http://www.alliedtelesis.co.nz/documentation/ar400/291/pdf/gre.pdf

- ✓ Routing:

KUROSE, James. Keith, Ross. "Computer Networking: A Top-Down Approach. Featuring the Internet", 6th edition. Addison-Wesley, 2013

http://www.cs.virginia.edu/~itlab/book/pdf/Ch3_v3.pdf

http://ptgmedia.pearsoncmg.com/images/9781587132063/samplechapter/1587132060_03.pdf

http://www.routeralley.com/guides/static_dynamic_routing.pdf

http://www.routeralley.com/guides/routing_protocol_comparison.pdf

http://ptgmedia.pearsoncmg.com/images/9781587132063/samplechapter/1587132060_03.pdf

http://www.cisco.com/c/en/us/td/docs/ios-xml/ios/iproute_bgp/configuration/15-mt/irg-15-mt-book.pdf

http://www.cisco.com/cisco/web/support/LA/7/76/76167_bgp-toc.html

http://www.cisco.com/cisco/web/support/LA/7/73/73125_bgpfaq_5816.html#

http://www.cisco.com/c/en/us/support/docs/ip/border-gateway-protocol-bgp/13753-25.html

https://www.ietf.org/rfc/rfc1771.txt

- ✓ VRRP:

https://tools.ietf.org/html/rfc3768

- ✓ EoIP:

http://wiki.mikrotik.com/wiki/Manual:Interface/EoIP

- ✓ Debian:

https://www.debian.org/index.es.html

https://www.debian.org/intro/about#what

https://www.debian.org/doc/manuals/project-history/index.es.html

https://www.debian.org/users/

https://www.debian.org/intro/why_debian

https://www.debian.org/doc/manuals/debian-faq/index.en.html#contents

- ✓ Cisco IOS:

http://www.cisco.com/c/en/us/products/ios-nx-os-*software*/ios-*software*-releases-listing.html

http://en.wikipedia.org/wiki/Cisco_IOS

- ✓ Openswan:

https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Security_Guide/openswan.html

http://en.wikipedia.org/wiki/Openswan

- ✓ Quagga:

http://www.nongnu.org/quagga/index.html

http://www.nongnu.org/quagga/docs/docs-info.html#Overview

http://www.nongnu.org/quagga/docs/docs-info.html#System-Architecture

- ✓ Pacemaker:

http://clusterlabs.org/

http://clusterlabs.org/doc/

http://www.linux-ha.org/wiki/STONITH

- ✓ Corosync:

http://corosync.github.io/corosync/

- ✓ Shell scripts:

http://www.freeos.com/guides/lsst/index.html

- ✓ Apache2:

http://httpd.apache.org/

http://wiki.apache.org/httpd/FAQ

- ✓ PostgreSQL:

http://www.postgresql.org.es/

http://www.postgresql.org.es/sobre_postgresql

- ✓ DRBD:

http://www.drbd.org/

http://swl.mes.edu.cu/DRBD/drbd-users-guide-1.3.4.pdf

- ✓ LVM:

https://www.centos.org/docs/5/pdf/Cluster_Logical_Volume_Manager.pdf

http://tldp.org/HOWTO/LVM-HOWTO/

- ✓ Certificados digitales:

http://www.uv.es/sto/articulos/BEI-2003-11/certificados_digitales.html#id2806966

Apéndice A.

Archivos de configuración del RouterI5.

Túnel IPsec.

En Openswan, para Debian 6, existen dos archivos importantes donde se definen los parámetros y características de una VPN. Éstos son: “ipsec.conf” e “ipsec.secrets”, ambos dentro del directorio /etc/ del sistema de archivos.

Por seguridad, no se revelará información confidencial como: direcciones IP de Internet, dominios de cifrado, algoritmos de *hashing* y cifrado y contraseñas.

- Archivo /etc/ipsec.conf:

```
version   2.0

# basic configuration

config setup

     nat_traversal=yes

     virtual_private=%v4:10.0.0.0/8,%v4:192.168.0.0/16,%v4:17
2.16.0.0/12

     oe=off

     protostack=klips


conn amphitryon-la_empresa

     auto=start

     authby=secret

     left=201.xxx.yyy.130

     leftnexthop=201.xxx.yyy.129

     leftsubnet=192.168.yyy.zzz/32

     leftsourceip=192.168.yyy.zzz

     right=50.xxx.yyy.238

     rightnexthop=201.xxx.yyy.130
```

```
        rightsubnet=172.16.206.0/27
        ike="[algoritmo_de_cifrado]-[hashing]-modp[wwww]"
        ikelifetime=28800s
        esp=[algoritmo_de_cifrado]-[hashing]
        keylife=3600s
        dpddelay=30
        dpdtimeout=120
        dpdaction=restart

conn carrier-la_empresa
        auto=start
        authby=secret
        left=201.xxx.yyy.130
        leftnexthop=201.xxx.yyy.129
        leftsubnet=192.168.yyy.zzz/32
        leftsourceip=192.168.yyy.zzz
        right=148.xxx.yyy.100
        rightnexthop=201.xxx.yyy.130
        rightsubnet=192.168.yyy.zzz/32
        ike="[algoritmo_de_cifrado]-[hashing]-modp[wwww]"
        ikelifetime=480m
        esp=[algoritmo_de_cifrado]-[hashing]
        keylife=60m
        dpddelay=30
        dpdtimeout=120
        dpdaction=restart
```

- Archivo /etc/ipsec.secrets:

```
201.xxx.yyy.130 50.xxx.yyy.238 : PSK "[frase_pre-compartida]"
201.xxx.yyy.130 148.xxx.yyy.100 : PSK "[frase_precompartida]"
```

Túnel GRE.

Los túneles GRE en Linux se construyen por medio de comandos del sistema ejecutados directamente en una terminal de texto. Pueden ser puestos en un *shell script* que se ejecute cada vez que el router encienda, para que el túnel exista siempre.

- Archivo /usr/local/bin/createTunneltoAmphitryon.sh:

```
#!/bin/bash

action=$1

directory="/usr/local/bin "

tunname="tunAmphitryon"
localip="192.168.yyy.zzz "
remoteip="172.16.206.13"
tunaddr="192.168.205.1/30"

ttl="255"

script_usage()
{
     echo -e "\nUsage:\t$0 {up|down|help}\n"
}
```

```
case $action in
     "up")
          ip tunnel add $tunname mode gre remote $remoteip
local $localip ttl $ttl
          ip link set $tunname up
          ip addr add $tunaddr dev $tunname
          ;;
     "down")
          $directory/removeTunnel.sh $tunname
          ;;
     "help")
          script_usage
          ;;
     *)
          script_usage
          exit 127
          ;;
esac

exit 0
```

- Archivo /usr/local/bin/createTunneltoCarrier.sh:

```
#!/bin/bash

action=$1
```

```
directory="/media/data/bin"


tunname="tun10"
localip="192.168.yyy.zzz"
remoteip="192.168.yyy.zzz"
tunaddr="192.168.41.6/30"


ttl="255"


script_usage()
{
     echo -e "\nUsage:\t$0 {up|down|help}\n"
}


case $action in
     "up")
          ip tunnel add $tunname mode gre remote $remoteip
local $localip ttl $ttl
          ip link set $tunname up
          ip link set $tunname mtu 1400
          ip addr add $tunaddr dev $tunname
          ;;
     "down")
          $directory/removeTunnel.sh $tunname
          ;;
     "help")
          script_usage
          ;;
```

```
     *)
          script_usage
          exit 127
          ;;
esac


exit 0
```

BGP.

Los archivos de configuración para los protocolos de *routing*, en Debian 6, se localizan en la ruta /etc/quagga/.

- Archivo /etc/quagga/bgpd.conf:

```
hostname RouterI5
password [contraseña]
!
router bgp 65210
bgp router-id 10.23.1.zzz
!
network 10.23.1.0/24
network 192.168.205.0/30
network 192.168.41.4/30
!
redistribute kernel
redistribute connected
redistribute static
!
```

```
neighbor 10.23.1.aaa remote-as 65210
neighbor 10.23.1.aaa route-map la_empresa out
!
neighbor 10.23.1.bbb remote-as 65210
neighbor 10.23.1.bbb route-map la_empresa out
!
neighbor 10.23.1.ccc remote-as 65210
neighbor 10.23.1.ccc route-map la_empresa out
!
neighbor 10.23.1.ddd remote-as 65210
neighbor 10.23.1.ddd route-map la_empresa out
!
neighbor 10.23.1.eee remote-as 65210
neighbor 10.23.1.eee route-map la_empresa out
!
neighbor 192.168.205.2 remote-as 65211
neighbor 192.168.205.2 weight 100
neighbor 192.168.205.2 route-map amphitryon out
!
neighbor 192.168.41.5 remote-as 65490
neighbor 192.168.41.5 weight 100
neighbor 192.168.41.5 route-map carrierin in
neighbor 192.168.41.5 route-map carrierout out
!
access-list 1 permit 10.23.0.0 0.0.0.255
access-list 2 permit 172.16.206.0 0.0.0.31
!
```

```
access-list 3 permit 10.23.1.0 0.0.0.255
access-list 3 permit 10.23.2.0 0.0.0.255
access-list 3 permit 10.23.3.0 0.0.0.255
access-list 3 permit 10.23.4.0 0.0.0.255
access-list 3 permit 10.23.5.0 0.0.0.255
access-list 3 permit 10.23.6.0 0.0.0.255
access-list 3 permit 10.23.7.0 0.0.0.255
!
access-list 4 permit 192.168.205.0 0.0.0.3
access-list 4 permit 192.168.yyy.zzz 0.0.0.0
access-list 4 permit 192.168.yyy.zzz 0.0.0.0
access-list 4 permit 192.168.41.4 0.0.0.3
access-list 4 permit 192.168.41.8 0.0.0.3
access-list 4 permit 192.168.yyy.zzz 0.0.0.0
access-list 4 permit 192.168.yyy.zzz 0.0.0.0
!
access-list 99 permit 10.xxx.yyy.0 0.0.1.255
access-list 99 permit 10.xxx.yyy.0 0.0.0.255
access-list 99 permit 10.xxx.yyy.0 0.0.7.255
access-list 99 permit 10.xxx.yyy.0 0.0.7.255
access-list 99 permit 10.xxx.yyy.0 0.0.15.255
access-list 99 permit 10.xxx.yyy.0 0.0.3.255
access-list 99 permit 10.xxx.yyy.0 0.0.1.255
access-list 99 permit 10.xxx.yyy.0 0.0.0.255
access-list 99 permit 172.xxx.yyy.0 0.0.1.255
access-list 99 permit 172.xxx.yyy.0 0.0.1.255
access-list 99 permit 172.xxx.yyy.0 0.0.7.255
```

```
access-list 99 permit 172.xxx.yyy.0 0.0.7.255
!
route-map la_empresa permit 10
 match ip address 1
 set ip next-hop 10.23.1.zzz
!
route-map la_empresa permit 20
 match ip address 2
 set ip next-hop 10.23.1.zzz
!
route-map la_empresa permit 30
 match ip address 3
 set ip next-hop 10.23.1.zzz
!
route-map la_empresa permit 40
 match ip address 4
 set ip next-hop 10.23.1.zzz
!
route-map la_empresa permit 60
 match ip address 99
 set ip next-hop 10.23.1.zzz
!
route-map amphitryon permit 10
 match ip address 3
 set ip next-hop 192.168.205.1
!
route-map amphitryon permit 40
```

```
 match ip address 99
 set ip next-hop 192.168.205.1
!
route-map carrierin permit 10
 match ip address 99
!
route-map carrierout permit 10
 match ip address 1
 set as-path prepend 65210
 set ip next-hop 192.168.41.6
!
route-map carrierout permit 20
 match ip address 3
 set as-path prepend 65210
 set ip next-hop 192.168.41.6
!
line vty
no login
```

Apéndice B

Archivos de configuración del RouterI4.

Túnel IPsec.

En Openswan, para Debian 6, existen dos archivos importantes donde se definen los parámetros y características de una VPN. Éstos son: “ipsec.conf” e “ipsec.secrets”, ambos dentro del directorio /etc/ del sistema de archivos.

Por seguridad, no se revelará información confidencial como: direcciones IP de Internet, dominios de cifrado, algoritmos de *hashing* y cifrado y contraseñas.

- Archivo /etc/ipsec.conf:

```
version   2.0

# basic configuration

config setup

     nat_traversal=yes

     virtual_private=%v4:10.0.0.0/8,%v4:192.168.0.0/16,%v4:17
2.16.0.0/12

     oe=off

     protostack=klips


conn amphitryon-la_empresa

     auto=start

     authby=secret

     left=200.xxx.yyy.218

     leftsubnet=192.168.yyy.zzz/32

     leftsourceip=192.168.yyy.zzz

     right=50.xxx.yyy.238

     rightsubnet=172.16.206.0/27

     ike="[algoritmo_de_cifrado]-[hashing]-modp[wwww]"
```

```
ikelifetime=28800s
esp=[algoritmo_de_cifrado]-[hashing]
keylife=3600s
dpddelay=30
dpdtimeout=120
dpdaction=restart
```

- Archivo /etc/ipsec.secrets:

```
200.xxx.yyy.218 50.xxx.yyy.238 : PSK "[frase_pre-compartida]"
```

Túnel GRE.

Los túneles GRE en Linux se construyen por medio de comandos del sistema ejecutados directamente en una terminal de texto. Pueden ser puestos en un *shell script* que se ejecute cada vez que el router encienda, para que el túnel exista siempre.

- Archivo /usr/local/bin/createTunneltoAmphitryon.sh:

```
#!/bin/bash

action=$1

directory="/usr/local/bin "

tunname="tunAmphitryon"
localip="192.168.yyy.zzz"
remoteip="172.16.206.13"
tunaddr="192.168.205.5/30"
```

```
ttl="255"


script_usage()
{
     echo -e "\nUsage:\t$0 {up|down|help}\n"
}


case $action in
     "up")
          ip tunnel add $tunname mode gre remote $remoteip
local $localip ttl $ttl
          ip link set $tunname up
          ip addr add $tunaddr dev $tunname
          ;;
     "down")
          $directory/removeTunnel.sh $tunname
          ;;
     "help")
          script_usage
          ;;
     *)
          script_usage
          exit 127
          ;;
esac


exit 0
```

BGP.

Los archivos de configuración para los protocolos de *routing*, en Debian 6, se localizan en la ruta /etc/quagga/.

- Archivo /etc/quagga/bgpd.conf:

```
hostname RouterI4
password [contraseña]
!
router bgp 65210
bgp router-id 10.23.1.zzz
!
network 10.23.1.0/24
network 192.168.205.4/30
!
redistribute kernel
redistribute connected
redistribute static
!
neighbor 10.23.1.aaa remote-as 65210
neighbor 10.23.1.aaa route-map la_empresa out
!
neighbor 10.23.1.bbb remote-as 65210
neighbor 10.23.1.bbb route-map la_empresa out
!
neighbor 10.23.1.ccc remote-as 65210
neighbor 10.23.1.ccc route-map la_empresa out
!
neighbor 10.23.1.ddd remote-as 65210
```

```
neighbor 10.23.1.ddd route-map la_empresa out
!
neighbor 10.23.1.eee remote-as 65210
neighbor 10.23.1.eee route-map la_empresa out
!
neighbor 192.168.205.6 remote-as 65211
neighbor 192.168.205.6 route-map amphitryon out
!
access-list 1 permit 10.23.0.0 0.0.0.255
!
access-list 2 permit 172.16.206.0 0.0.0.31
!
access-list 3 permit 10.23.1.0 0.0.0.255
access-list 3 permit 10.23.2.0 0.0.0.255
access-list 3 permit 10.23.3.0 0.0.0.255
access-list 3 permit 10.23.4.0 0.0.0.255
access-list 3 permit 10.23.5.0 0.0.0.255
access-list 3 permit 10.23.6.0 0.0.0.255
access-list 3 permit 10.23.7.0 0.0.0.255
!
access-list 4 permit 192.168.205.4 0.0.0.3
access-list 4 permit 192.168.yyy.zzz 0.0.0.0
access-list 4 permit 192.168.41.8 0.0.0.3
!
access-list 99 permit 10.xxx.yyy.0 0.0.1.255
access-list 99 permit 10.xxx.yyy.0 0.0.0.255
access-list 99 permit 10.xxx.yyy.0 0.0.7.255
```

```
access-list 99 permit 10.xxx.yyy.0 0.0.7.255
access-list 99 permit 10.xxx.yyy.0 0.0.15.255
access-list 99 permit 10.xxx.yyy.0 0.0.3.255
access-list 99 permit 10.xxx.yyy.0 0.0.1.255
access-list 99 permit 10.xxx.yyy.0 0.0.0.255
access-list 99 permit 172.xxx.yyy.0 0.0.1.255
access-list 99 permit 172.xxx.yyy.0 0.0.1.255
access-list 99 permit 172.xxx.yyy.0 0.0.7.255
access-list 99 permit 172.xxx.yyy.0 0.0.7.255
!
route-map la_empresa permit 10
 match ip address 1
 set ip next-hop 10.23.1.zzz
!
route-map la_empresa permit 20
 match ip address 2
 set ip next-hop 10.23.1.zzz
!
route-map la_empresa permit 30
 match ip address 3
 set ip next-hop 10.23.1.zzz
!
route-map la_empresa permit 40
 match ip address 4
 set ip next-hop 10.23.1.zzz
!
route-map la_empresa permit 50
```

```
 match ip address 99
 set ip next-hop 10.23.1.zzz
!
route-map amphitryon permit 10
 match ip address 3
 set as-path prepend 65210
 set ip next-hop 192.168.205.5
!
route-map amphitryon permit 40
 match ip address 99
 set ip next-hop 192.168.205.5
!
line vty
no login
```

Apéndice C.

Configuración del Cisco ISR 1921 (*site* Ciudad de México).

BGP.

El siguiente fragmento de la configuración del *router* para BGP fue tomado directamente del dispositivo (los prefijos de las redes involucradas son confidenciales):

```
router bgp 65210
 bgp router-id 10.23.1.zzz
 bgp log-neighbor-changes
 network 10.23.1.0 mask 255.255.255.0
 network 192.168.yyy.zzz mask 255.255.255.240
 redistribute connected
 redistribute static
 neighbor 10.23.1.aaa remote-as 65210
 neighbor 10.23.1.aaa route-map la_empresa out
 neighbor 10.23.1.bbb remote-as 65210
 neighbor 10.23.1.bbb route-map la_empresa out
 neighbor 10.23.1.ccc remote-as 65210
 neighbor 10.23.1.ccc route-map la_empresa out
 neighbor 10.23.1.ddd remote-as 65210
 neighbor 10.23.1.ddd route-map la_empresa out
 neighbor 10.23.1.eee remote-as 65210
 neighbor 10.23.1.eee route-map la_empresa out
 neighbor 192.168.yyy.zzz remote-as 65490
 neighbor 192.168.yyy.zzz weight 100
 neighbor 192.168.yyy.zzz route-map carrierin in
 neighbor 192.168.yyy.zzz route-map carrierout out
```

```
!
access-list 1 permit 10.23.0.0 0.0.0.255
access-list 2 permit 10.23.1.0 0.0.0.255
access-list 2 permit 10.23.2.0 0.0.0.255
access-list 2 permit 10.23.3.0 0.0.0.255
access-list 2 permit 10.23.4.0 0.0.0.255
access-list 2 permit 10.23.5.0 0.0.0.255
access-list 2 permit 10.23.6.0 0.0.0.255
access-list 2 permit 10.23.7.0 0.0.0.255
access-list 99 permit 192.168.yyy.zzz 0.0.0.15
access-list 99 permit 10.xxx.yyy.0 0.0.1.255
access-list 99 permit 10.xxx.yyy.0 0.0.0.255
access-list 99 permit 10.xxx.yyy.0 0.0.7.255
access-list 99 permit 10.xxx.yyy.0 0.0.7.255
access-list 99 permit 10.xxx.yyy.0 0.0.15.255
access-list 99 permit 10.xxx.yyy.0 0.0.3.255
access-list 99 permit 10.xxx.yyy.0 0.0.1.255
access-list 99 permit 10.xxx.yyy.0 0.0.0.255
access-list 99 permit 172.xxx.yyy.0 0.0.1.255
access-list 99 permit 172.xxx.yyy.0 0.0.1.255
access-list 99 permit 172.xxx.yyy.0 0.0.7.255
access-list 99 permit 172.xxx.yyy.0 0.0.7.255
!
route-map la_empresa permit 10
 match ip address 2 99
 set ip next-hop 10.23.1.zzz
!
```

```
route-map carrierin permit 10
 match ip address 99
!
route-map carrierout permit 10
 match ip address 1 2
 set ip next-hop 192.168.yyy.0
```

Apéndice D

Configuración del Cisco ISR 1921 (*site* Chicago).

Túneles GRE.

Los túneles GRE creados en el dispositivo tienen la siguiente configuración (los prefijos de las redes involucradas son confidenciales):

```
interface Tunnel0
 description GRE Tunnel to La Empresa Router VPN 1.
 ip address 192.168.205.2 255.255.255.252
 tunnel source 172.16.206.13
 tunnel destination 192.168.yyy.zzz
!
interface Tunnel1
 description GRE Tunnel to La Empresa Router VPN 2.
 ip address 192.168.205.6 255.255.255.252
 tunnel source 172.16.206.13
 tunnel destination 192.168.yyy.zzz
!
interface Tunnel11
 description GRE Tunnel to Carrier.
 ip address 192.168.41.10 255.255.255.252
 ip access-group 99 in
 ip mtu 1400
 ip tcp adjust-mss 1360
 tunnel source 172.16.206.13
 tunnel destination 192.168.yyy.zzz
```

VRRP.

Los comandos para activar VRRP en cada uno de los *routers* se incluyen a continuación.

- Router 1:

```
vrrp 10 description Virtual IP for GRE tunnels.
 vrrp 10 ip 172.16.206.13
 vrrp 10 priority 250
```

- Router 2:

```
vrrp 10 description Virtual IP for GRE tunnels.
 vrrp 10 ip 172.16.206.13
 vrrp 10 priority 200
```

BGP.

El siguiente fragmento de la configuración del *router* para BGP fue tomado directamente del dispositivo (los prefijos de las redes involucradas son confidenciales):

```
router bgp 65211
 bgp router-id 172.16.206.zzz
 bgp log-neighbor-changes
 network 172.16.206.0 mask 255.255.255.224
 network 192.168.41.8 mask 255.255.255.252
 network 192.168.205.0 mask 255.255.255.252
 network 192.168.205.4 mask 255.255.255.252
 redistribute connected
 redistribute static
 neighbor 172.16.206.zzz remote-as 65211
 neighbor 172.16.206.zzz route-map amphitryon out
```

```
 neighbor 192.168.41.9 remote-as 65490
 neighbor 192.168.41.9 weight 100
 neighbor 192.168.41.9 route-map carrierin in
 neighbor 192.168.41.9 route-map carrierout out
 neighbor 192.168.205.1 remote-as 65210
 neighbor 192.168.205.1 route-map la_empresauno out
 neighbor 192.168.205.5 remote-as 65210
 neighbor 192.168.205.5 route-map la_empresados out
!
access-list 1 permit 10.23.0.0 0.0.0.255
access-list 2 permit 10.23.1.0 0.0.0.255
access-list 2 permit 10.23.2.0 0.0.0.255
access-list 2 permit 10.23.3.0 0.0.0.255
access-list 2 permit 10.23.4.0 0.0.0.255
access-list 2 permit 10.23.5.0 0.0.0.255
access-list 2 permit 10.23.6.0 0.0.0.255
access-list 2 permit 10.23.7.0 0.0.0.255
access-list 99 permit 192.168.41.8 0.0.0.3
access-list 99 permit 10.xxx.yyy.0 0.0.1.255
access-list 99 permit 10.xxx.yyy.0 0.0.0.255
access-list 99 permit 10.xxx.yyy.0 0.0.7.255
access-list 99 permit 10.xxx.yyy.0 0.0.7.255
access-list 99 permit 172.xxx.yyy.0 0.0.1.255
access-list 99 permit 172.xxx.yyy.0 0.0.1.255
access-list 99 permit 172.xxx.yyy.0 0.0.7.255
access-list 99 permit 172.xxx.yyy.0 0.0.7.255
access-list 99 permit 10.xxx.yyy.0 0.0.15.255
```

```
access-list 99 permit 10.xxx.yyy.0 0.0.3.255
access-list 99 permit 10.xxx.yyy.0 0.0.1.255
access-list 99 permit 10.xxx.yyy.0 0.0.0.255
!
route-map carrierin permit 10
 match ip address 99
!
route-map la_empresauno permit 10
 match ip address 1 99
 set ip next-hop 192.168.205.2
!
route-map carrierout permit 10
 match ip address 1
 set ip next-hop 192.168.41.10
!
route-map carrierout permit 20
 match ip address 2
 set as-path prepend 65211
 set ip next-hop 192.168.41.10
!
route-map la_empresados permit 10
 match ip address 1 99
 set ip next-hop 192.168.205.6
!
route-map amphitryon permit 10
 match ip address 2 99
 set ip next-hop 172.16.206.zzz
```

Apéndice E.

Configuración del *cluster* web.

Corosync.

Para que pueda darse la comunicación entre los nodos dentro del *cluster*, se decidió manipular las opciones de corosync para una transmisión *unicast* con UDP y posteriormente habilitarlo para que se ejecute en automático como un demonio cada vez que se reinicie el Sistema Operativo. Nota: las direcciones IP involucradas son confidenciales.

- Archivo /etc/corosync/corosync.conf:

```
compatibility: whitetank


totem {
     version: 2
     secauth: off
     interface {
          member {
               memberaddr: 10.23.0.xxx
          }
          member {
               memberaddr: 10.23.0.yyy
          }
          ringnumber: 0
          bindnetaddr: 10.23.0.0
          mcastport: 5405
          ttl: 1
     }
     transport: udpu
```

```
}

logging {
     fileline: off
     to_logfile: no
     to_syslog: yes
     debug: on
     debug: off
     timestamp: on
     logger_subsys {
          subsys: AMF
          debug: off
     }
}

service {
        # Load the Pacemaker Cluster Resource Manager
        name: pacemaker
        ver: 0
}
```

- Archivo /etc/default/corosync:

```
# start corosync at boot [yes|no]
START=yes
```

Pacemaker.

Los comandos de Pacemaker que deben ejecutarse en terminal para la configuración de los recursos Apache2, dirección IP virtual y *ping* se encuentran en las siguientes líneas. Las instrucciones deben aplicarse únicamente en uno de los nodos. Algunos datos importantes se han ocultado por motivos de seguridad y privacidad.

1) Deshabilitar STONITH y validación de quorum (ya que sólo son 2 nodos). Establecer un valor de *stickiness* para el nodo que posea los recursos:

```
crm configure property stonith-enabled=false

crm configure property no-quorum-policy=ignore

crm configure rsc_defaults resource-stickiness=100
```

2) Agregar una dirección IP virtual administrada por Pacemaker para el *failover* entre los nodos:

```
crm configure primitive virtual-ip ocf:heartbeat:IPaddr2
params ip="10.23.0.zzz" cidr_netmask="27" \

     op monitor interval="30s"
```

3) Agregar a Apache2 como recurso administrado por Pacemaker:

```
crm configure primitive apache-svc ocf:la_empresa:apache2
params http_url="www.clusterla_empresa.org" \

     tries=2 timeout_response=3 sleep_time="10" op start
timeout="90s" op stop timeout="100s" \

     op monitor interval="30s" timeout="20s"
```

4) Asociar los recursos de Pacemaker:

```
crm configure colocation apache-svc_and_virtual-ip INFINITY:
apache-svc virtual-ip
```

5) Especificar el orden en el que inician los recursos en el nodo:

```
crm configure order apache-svc_after_virtual-ip mandatory:
virtual-ip apache-svc
```

6) Forzar la asignación de recursos a un nodo preferencial:

```
crm configure location apache-svc_in_524740-web1 apache-svc
50: 524740-web1
```

7) Agregar el recurso de *ping* hacia un par de direcciones para verificar la comunicación con la red:

```
crm configure primitive KeepAlive ocf:pacemaker:ping params
name=ping dampen=5s \
     multiplier=100 host_list="10.23.0.xxx\ 10.23.0.yyy" op
start timeout="60s" \
     op monitor interval="30s" timeout="60s"
```

8) Clonar el recurso de *ping* para que permanezca activo en ambos nodos. Consideramos favorable que ambos nodos verifiquen persistentemente su comunicación con la red, ya que no tiene caso migrar los recursos a un nodo que no puede transmitir hacia el exterior:

```
crm configure clone KAclone KeepAlive
location KAwithIP virtual-ip \
     rule 50: #uname eq "524740-web1" \
     rule ping: defined ping
```

Resource agent de Apache2.

Por último, el *resource agent* encargado de controlar el servicio (ocf:la_empresa:apache2) es un *script* programado y personalizado de acuerdo a nuestro criterio, resultado de mi experiencia con el *software* para arrancar, detener y validar si el demonio de Apache2 está funcionando correctamente. A continuación se muestra el *shell script* completo:

```
#!/bin/sh


# High-Availability Apache2 OCF resource agent for Linux
Debian® Squeeze.

# Description: Start, stop, check status and verify
environment of apache2 daemon in a web server.

# Author: Saúl Cortés Riquelme.

# License:     GNU General Public License v2 (GPLv2).

# Derived in part from: Apache2 OCF resource agent of Florian
Knauf.

# OCF parameters used:

# OCF_RESKEY_binfile          default: /usr/sbin/apache2ctl

# OCF_RESKEY_pidfile          default: /var/run/apache2.pid

# OCF_RESKEY_logfile          default:
/var/log/pacemaker/apache2_ra

# OCF_RESKEY_http_url         default: localhost

# OCF_RESKEY_https_url        default: localhost

# OCF_RESKEY_tries       default: 1

# OCF_RESKEY_sleep_time       default: 5

# OCF_RESKEY_timeout_response default: 1

# Performance of this resource agent was tested in a Debian®
Squeeze O. S. with apache2 2.2.16 version environment.

# To test in another version, I recommended a previous
assessment of its operation.
```

```
## Initializing resource agent script.

. ${OCF_ROOT}/resource.d/heartbeat/.ocf-shellfuncs


## Exporting environment variables.

[ -n "$OCF_RESKEY_binfile" ] && export APACHE_BINFILE="-f
$OCF_RESKEY_binfile"

[ -n "$OCF_RESKEY_pidfile" ] && export APACHE_PIDFILE="-f
$OCF_RESKEY_pidfile"

[ -n "$OCF_RESKEY_logfile" ] && export APACHE_LOGFILE="-f
$OCF_RESKEY_logfile"

[ -n "$OCF_RESKEY_http_url"  ] && export APACHE_HTTPURL="-f
$OCF_RESKEY_http_url"

[ -n "$OCF_RESKEY_https_url"  ] && export APACHE_HTTPSURL="-f
$OCF_RESKEY_https_url"

[ -n "$OCF_RESKEY_tries" ] && export APACHE_RETRIES="-f
$OCF_RESKEY_tries"

[ -n "$OCF_RESKEY_sleep_time" ] && export APACHE_SLEEPTIME="-
f $OCF_RESKEY_sleep_time"

[ -n "$OCF_RESKEY_timeout_response" ] && export
APACHE_TIMEOUT="-f $OCF_RESKEY_timeout_response"


## Reading action received.

command="$1"


## Declaring global variables.

binfile="$OCF_RESKEY_binfile"

pidfile="$OCF_RESKEY_pidfile"

logfile="$OCF_RESKEY_logfile"

http_url="$OCF_RESKEY_http_url"

https_url="$OCF_RESKEY_https_url"

tries="$OCF_RESKEY_tries"
```

```
sleep_time="$OCF_RESKEY_sleep_time"
timeout_response="$OCF_RESKEY_timeout_response"
wget_cmd='/usr/bin/wget'
indexfiles='/var/lib/heartbeat/cores/root/index.html*'

## Assigning default values to unspecified variables.
if [ -z "$binfile" ]
     then
          binfile='/usr/sbin/apache2ctl'
fi

if [ -z "$pidfile" ]
     then
          pidfile='/var/run/apache2.pid'
fi

if [ -z "$logfile" ]
     then
          logfile='/var/log/pacemaker/apache2_ra'
fi

if [ -z "$http_url" ]
     then
          http_url='localhost'
fi

if [ -z "$https_url" ]
```

```
     then
          https_url='localhost'
fi

if [ -z "$tries" ]
     then
          tries='1'
fi

if [ -z "$sleep_time" ]
     then
          sleep_time='5'
fi

if [ -z "$timeout_response" ]
     then
          timeout_response='1'
fi

## Defining composite variables.
http_verify="$wget_cmd -t $tries -T $timeout_response
http://$http_url/"
https_verify="$wget_cmd -t $tries -T $timeout_response --no-
check-certificate https://$https_url/"

## Usage function.
usage()
{
```

```
     echo "\nUsage:\t$0 {start|stop|monitor|meta-
data|validate-all|usage}\n"

}


## Meta-data function (show XML meta-data).

metadata_apache()

{

     cat <<END

<?xml version="1.0"?>

<!DOCTYPE resource-agent SYSTEM "ra-api-1.dtd">

<resource-agent name="apache2">

<version>1.0</version>


<longdesc lang="en">

     Resource agent that controls apache2 daemon in a Debian®
O.S. environment.

</longdesc>


<shortdesc lang="en">

     R. A. for apache2 web server.

</shortdesc>


<parameters>

<parameter name="binfile" unique="0" required="0">

     <longdesc lang="en">

          Full path to script file that controls the
launching and halting of apache2 process.

     </longdesc>

     <shortdesc lang="en">apache2 binary file</shortdesc>
```

```
     <content type="string" default="/usr/sbin/apache2ctl"/>
</parameter>
<parameter name="pidfile" required="0" unique="1">
     <longdesc lang="en">
          Apache2 PID file that exist while apache2 daemon is
running. This file never will exist if apache2 is stopped.
Please provide full path for this file.
     </longdesc>
     <shortdesc lang="en">
          Full path of apache2 PID file.
          </shortdesc>
     <content type="string" default="/var/run/apache2.pid"/>
</parameter>
<parameter name="logfile">
     <longdesc lang="en">
          The full path for Apache2 Resource Agent log file,
where principal events will be reported.
     </longdesc>
     <shortdesc lang="en">
          Full path of apache2 Resource Agent log file.
     </shortdesc>
     <content type="string"
default="/var/log/pacemaker/apache2_ra"/>
</parameter>
<parameter name="http_url">
     <longdesc lang="en">
          The URL or IP address of the apache2 HTTP service.
This parameter will be used for a wget test.
     </longdesc>
     <shortdesc lang="en">
```

```
          URL to test for HTTP.
     </shortdesc>
     <content type="string" default="localhost"/>
</parameter>
<parameter name="https_url">
     <longdesc lang="en">
          The URL of the apache2 HTTPS service. This
parameter will be used for a wget test.
     </longdesc>
     <shortdesc lang="en">
          URL to test for HTTPS.
     </shortdesc>
     <content type="string" default="localhost"/>
</parameter>
<parameter name="tries">
     <longdesc lang="en">
          The number of tries for obtain a response of the
URL (see url parameters) web page. Applies for both HTTP and
HTTPS.
     </longdesc>
     <shortdesc lang="en">
          Retries for HTTPS response
     </shortdesc>
     <content type="integer" default="1"/>
</parameter>
<parameter name="sleep_time">
     <longdesc lang="en">
          The enough time, in seconds, to wait before
consider that Apache2 is started, stopped or killed after
such instruction.
```

```
     </longdesc>
     <shortdesc lang="en">
          Time to wait for start or stop Apache2.
     </shortdesc>
     <content type="integer" default="5"/>
</parameter>
<parameter name="timeout_response">
     <longdesc lang="en">
          The timeout, in seconds, to wait for a response of
the URL (see url parameters) web page. Applies for both HTTP
and HTTPS.
     </longdesc>
     <shortdesc lang="en">
          Number of seconds to wait for localhost web
response
     </shortdesc>
     <content type="integer" default="1"/>
</parameter>
</parameters>


<actions>
<action name="start"             timeout="90" />
<action name="stop"              timeout="100" />
<action name="monitor" depth="0" timeout="20" interval="10"
start-delay="1m" />
<action name="meta-data"         timeout="5" />
<action name="validate-all"      timeout="5" />
</actions>
```

```
</resource-agent>
END
}

## Log function.
log_action()
{
     stage="$1"
     logtext="$2"
     now=`date +%F\ %T.%N`

     echo -e "$now\t$stage:\t$logtext" >> $logfile
}

## Start/stop generic function.
exec_apache2ctl()
{
     action="$1"
     log_action 'EXECUTION' "apache2 $action."

     $binfile $action
     apache2ctl_result="$?"

     if [ "$apache2ctl_result" != "$OCF_SUCCESS" ]
          then
               log_action 'EXECUTION' "apache2 $action was
failed.\t(Code: $apache2ctl_result)"
               return $apache2ctl_result
```

```
        fi

        sleep $sleep_time
        return $OCF_SUCCESS
}



## Start function.
start_apache()
{
        log_action 'START' '\tStarting apache2...'
        log_action 'START' '\tReceived parameters:'
                log_action '\t:' "\t\t\t\tbinfile     =  $binfile"
                log_action '\t:' "\t\t\t\tpidfile     =  $pidfile"
                log_action '\t:' "\t\t\t\tlogfile     =  $logfile"
                log_action '\t:' "\t\t\t\thttp_url    =  $http_url"
                log_action '\t:' "\t\t\t\thttps_url   =
$https_url"
                log_action '\t:' "\t\t\t\ttries       =  $tries"
                log_action '\t:' "\t\t\t\tsleep_time  =
$sleep_time [s]"
                log_action '\t:' "\t\t\t\ttimeout_response  =
$timeout_response [s]"

        exec_apache2ctl start
        start_result="$?"

        if [ "$start_result" != "$OCF_SUCCESS" ]
```

```
          then
               log_action 'START' "\tapache2 can not be
started.\t(Code: $start_result)"
               kill_apache
               return $start_result
     fi

     monitor_apache
     start_status="$?"

     if [ "$start_status" != "$OCF_SUCCESS" ]
          then
               log_action 'START' "\tapache2 is not running
after start action.\t(Code: $start_status)"
               kill_apache
               return $start_status
     fi

     log_action 'START' '\tapache2 starts successful.\n'
     return $OCF_SUCCESS
}

## Stop function.
stop_apache()
{
     log_action 'STOP' '\tStopping apache2...'

     exec_apache2ctl stop
```

```
    stop_result="$?"

    if [ "$stop_result" != "$OCF_SUCCESS" ]
        then
            kill_apache
    fi

    monitor_apache
    stop_status="$?"

    if [ "$stop_status" != "$OCF_NOT_RUNNING" ]
        then
            log_action 'STOP' "\tapache2 can not be
stopped.\t(Code: $stop_status)"
            kill_apache
            kill_result="$?"
            if [ "$kill_result" != "$OCF_SUCCESS" ]
                then
                    log_action 'STOP' "\tapache2 stop
action was failed.\t(Code: $kill_result)"
                    return $kill_result
            fi
    fi

    log_action 'STOP' '\tapache2 stops successful.\n'
    return $OCF_SUCCESS
}
```

```
## Kill function.
kill_apache()
{
     log_action 'KILL' '\tKilling apache2...'

     kill -TERM $(pidof apache2) > /dev/null 2>&1
     sleep $sleep_time

     pidof apache2 > /dev/null
     pidof_result="$?"

     if [ "$pidof_result" != "$OCF_SUCCESS" ]
          then
               log_action 'KILL' '\tapache2 killed by TERM
signal.'
               return $OCF_SUCCESS
     fi

     kill -9 $(pidof apache2) > /dev/null 2>&1
     sleep $sleep_time

     pidof apache2 > /dev/null
     pidof_result="$?"

     if [ "$pidof_result" != "$OCF_SUCCESS" ]
          then
               log_action 'KILL' '\tapache2 killed by KILL
signal.'
```

```
                return $OCF_SUCCESS
        fi

        log_action 'KILL' '\tattempt to kill apache2 was
failed.'
        return $OCF_ERR_UNIMPLEMENTED
}

## Monitor function.
monitor_apache()
{
        log_action 'MONITOR' 'apache2 monitoring...'

        if [ -e "$pidfile" ]
                then
                        pidfile_status="$OCF_SUCCESS"
                else
                        log_action 'MONITOR' 'apache2 PID file does
not exist!'
                        pidfile_status="$OCF_NOT_RUNNING"
        fi

        pidof apache2 > /dev/null
        pidapache_status="$?"

        if [ "$pidapache_status" != "$OCF_SUCCESS" ]
                then
                        log_action 'MONITOR' 'apache2 daemon not has
PID!'
```

```
     fi


     $http_verify > /dev/null
     http_status="$?"


     if [ "$http_status" != "$OCF_SUCCESS" ]
          then
               log_action 'MONITOR' "HTTP service has
failed!\t(Code: $http_status)"
     fi


     $https_verify > /dev/null
     https_status="$?"


     if [ "$https_status" != "$OCF_SUCCESS" ]
          then
               log_action 'MONITOR' "HTTPS service has
failed!\t(Code: $https_status)"
     fi


     if [ "$pidfile_status" = "$OCF_SUCCESS" ] && [
"$pidapache_status" = "$OCF_SUCCESS" ] && [ "$http_status" =
"$OCF_SUCCESS" ] && [ "$https_status" = "$OCF_SUCCESS" ]
          then
               log_action 'MONITOR' 'apache2 is running OK.'
               rm $indexfiles > /dev/null 2>&1
               return $OCF_SUCCESS
     elif [ "$pidfile_status" != "$OCF_SUCCESS" ] && [
"$pidapache_status" != "$OCF_SUCCESS" ] && [ "$http_status"
!= "$OCF_SUCCESS" ] && [ "$https_status" != "$OCF_SUCCESS" ]
```

```
          then
               log_action 'MONITOR' 'apache2 is not running.'
               rm $indexfiles > /dev/null 2>&1
               return $OCF_NOT_RUNNING
          else
               log_action 'MONITOR' 'apache2 is running
partially.'
               rm $indexfiles > /dev/null 2>&1
               return $OCF_ERR_GENERIC
     fi
}

## Validate binaries function.
validate_all_apache()
{
     have_binary "$binfile" ||
     return $OCF_ERR_INSTALLED

     if [ ! -x "$binfile" ]
          then
               return $OCF_ERR_PERM
     fi

     output=`$binfile configtest 2>&1` || {
     ocf_log err "$output"
     return $OCF_ERR_CONFIGURED
     }
```

```
     return $OCF_SUCCESS
}



## MAIN.


if [ "$#" != '1' ]
     then
          log_action 'USAGE' "\tThis Resource Agent only can
receive one argument (args received: $#).\n"
          usage
          exit $OCF_ERR_ARGS
fi


validate_all_apache
validate_status="$?"


if [ "$validate_status" != "$OCF_SUCCESS" ]
     then
          log_action 'VERIFY' "\tValidation function has
returned an error.\t(Code: $validate_status)\n"
          exit $validate_status
fi


case "$command" in
     start)
          start_apache        ;;
     stop)
```

```
        stop_apache         ;;
    monitor)
        monitor_apache      ;;
    validate-all)
        validate_all_apache
        exit $?             ;;
    meta-data)
        metadata_apache
        exit $OCF_SUCCESS   ;;
    usage)
        usage
        exit $OCF_SUCCESS   ;;
    *)
        log_action 'ERROR' "\tThe action received ($action)
is not implemented on this Resource Agent script!\n"
        usage
        exit $OCF_ERR_UNIMPLEMENTED   ;;
esac
```

Apéndice F.

Configuración del *cluster* de base de datos.

Corosync.

Para que pueda darse la comunicación entre los nodos dentro del *cluster*, se decidió manipular las opciones de corosync para una transmisión *unicast* con UDP y posteriormente habilitarlo para que se ejecute en automático como un demonio cada vez que se reinicie el Sistema Operativo. Nota: las direcciones IP involucradas son confidenciales.

- Archivo /etc/corosync/corosync.conf:

```
compatibility: whitetank


totem {
     version: 2
     secauth: off
     interface {
          member {
               memberaddr: 10.23.0.xxx
          }
          member {
               memberaddr: 10.23.0.yyy
          }
          ringnumber: 0
          bindnetaddr: 10.23.0.0
          mcastport: 5405
          ttl: 1
     }
     transport: udpu
```

```
}

logging {
     fileline: off
     to_logfile: no
     to_syslog: yes
     debug: on
     debug: off
     timestamp: on
     logger_subsys {
          subsys: AMF
          debug: off
     }
}

service {
        # Load the Pacemaker Cluster Resource Manager
        name: pacemaker
        ver: 0
}
```

- Archivo /etc/default/corosync:

```
# start corosync at boot [yes|no]
START=yes
```

Pacemaker.

Los *clusters* de base de datos generalmente tienen una configuración más compleja debido a que deben sincronizarse los diferentes discos duros en donde se almacena la información y se debe ser muy cuidadoso para impedir que ésta se corrompa.

Los pasos y los comandos que seguí para la creación del *cluster* de base de datos se explican a continuación.

Procedimiento para configurar DRBD, respaldando los datos de PostgreSQL en dos nodos.

NOTA: Los símbolos "++" indican que la operación debe ser realizada en ambos nodos, mientras que "+" significa que la instrucción debe ejecutarse sólo en el nodo principal (a menos que se indique otra cosa).

1) Definir un filtro adecuado para LVM y deshabilitar memoria caché en /etc/lvm/lvm.conf: (++)

```
filter = [ "r|/dev/sda|", "r|/dev/disk/*|",
"r|/dev/block/*|", "a|.*|" ]

write_cache_state = 0
```

2) Regenerar el mapa de dispositivos detectados por el kernel: (++)

```
update-initramfs -u
```

3) Crear o modificar el archivo /etc/drbd.d/postgresql.res con la siguiente información: (++)

```
resource postgresql {
     device /dev/drbd0;
     meta-disk internal;
     syncer {
          rate 300K;
     }
     on 524743-db1 {
          address 10.23.0.xxx:7788;
```

```
        disk /dev/sda3;
    }
    on 524746-db2 {
        address 10.23.0.yyy:7788;
        disk /dev/sda4;
    }
}
```

4) Deshabilitar el recurso DRBD como servicio de arranque: (++)

```
chkconfig drbd off
```

5) Crear un nuevo recurso de DRBD: (++)

```
drbdadm create-md postgresql
```

6) Habilitar el nuevo recurso de DRBD: (++)

```
drbdadm up postgresql
```

7) Arrancar manualmente el servicio de DRBD: (++)

```
service drbd start
```

8) Indicar que el nodo primario escribirá su información en el otro secundario: (+)

```
drbdadm -- --overwrite-data-of-peer primary postgresql
```

9) Habilitar el recurso “lvm2” como servicio de arranque: (++)

```
chkconfig lvm2 off
```

10) Crear un volumen fisico: (+)

```
pvcreate /dev/drbd0
```

11) Crear un grupo de volúmenes: (+)

```
vgcreate postgresqlVG /dev/drbd0
```

12) Crear un volumen lógico de 350 [GB]: (+)

```
lvcreate -L 350G -n postgresqlLV postgresqlVG
```

13) Generar una carpeta para el punto de montaje: (++)

```
mkdir -p /mnt/drbd/postgresql/
```

14) Formatear el volumen lógico: (+)

```
mkfs.xfs -d agcount=8 /dev/postgresqlVG/postgresqlLV
```

15) Montar el volumen lógico en la carpeta generada en el punto 12: (+)

```
mount -t xfs -o noatime,nodiratime,attr2
/dev/postgresqlVG/postgresqlLV /mnt/drbd/postgresql
```

16) Cambiar el propietario y el grupo del punto de montaje: (+)

```
chown postgres:postgres /mnt/drbd/postgresql
```

17) Modificar los permisos del punto de montaje: (+)

```
chmod 0700 /mnt/drbd/postgresql
```

18) Bloquear el *cluster* principal del proceso de postgresql: (++)

```
pg_dropcluster 8.4 main
```

19) Crear un nuevo *cluster* del proceso de postgresql en la nueva carpeta: (++)

```
pg_createcluster -d /mnt/drbd/postgresql -s
/var/run/postgresql 8.4 postgresqlha
```

20) Eliminar el contenido que pueda haber en la nueva carpeta (IMPORTANTE: sólo en el nodo secundario):

```
rm -Rf /mnt/drbd/postgresql/*
```

21) Deshabilitar el recurso DRBD como servicio de arranque: (++)

```
chkconfig postgresql off
```

22) Arrancar postgresql: (+)

```
/etc/init.d/postgresql start
```

Procedimiento para configurar Pacemaker y la lógica del *cluster*.

Las siguientes instrucciones únicamente deben ejecutarse en uno de los nodos, cualquiera de ellos.

1) Deshabilitar STONITH y validación de quorum (ya que sólo son 2 nodos). Establecer un valor de *stickiness* para el nodo que posea los recursos.

```
crm configure property stonith-enabled=false

crm configure property no-quorum-policy=ignore

crm configure property default-resource-stickiness="150"
```

2) Agregar a DRBD como recurso administrado por Pacemaker y definir las instancias maestro/esclavo:

```
crm configure primitive drbd_postgresql ocf:linbit:drbd
params drbd_resource="postgresql" op start timeout="240" op
stop timeout="120" op monitor interval="30"
```

```
crm configure ms master_drbd_postgresql drbd_postgresql meta
master-max="1" master-node-max="1" clone-max="2" clone-node-
max="1" notify="true"
```

3) Agregar LVM y Filesystem como recursos administrados por Pacemaker:

```
crm configure primitive lvm_postgresql ocf:heartbeat:LVM
params volgrpname="postgresqlVG" op start timeout="30" op
stop timeout="30"
```

```
crm configure primitive hanfs-svc ocf:heartbeat:Filesystem
params device="/dev/postgresqlVG/postgresqlLV"
directory="/mnt/drbd/postgresql" options="noatime,nodiratime"
fstype="xfs" op start timeout="60" op stop timeout="120"
```

4) Agregar a PostgreSQL como recurso administrado por Pacemaker:

```
crm configure primitive pgsql-svc ocf:la_empresa:postgresql
op start timeout="120s" op stop timeout="120s" op monitor
interval="60s" timeout="30s"
```

5) Agregar una dirección IP virtual administrada por Pacemaker para el *failover* entre los nodos:

```
crm configure primitive ip_postgresql ocf:heartbeat:IPaddr2
params ip="10.23.0.zzz" cidr_netmask="27" op monitor
interval="30s"
```

6) Definir un grupo que englobe a todos los recursos de Pacemaker:

```
crm configure group server_postgresql lvm_postgresql hanfs-
svc pgsql-svc ip_postgresql
```

7) Asociar los recursos de Pacemaker indicados:

```
crm configure colocation serverpg_with_masterdrbd inf:
server_postgresql master_drbd_postgresql:Master
```

8) Especificar el orden en el que inician los recursos en el nodo:

```
crm configure order order_postgresql inf:
master_drbd_postgresql:promote server_postgresql:start
```

9) Forzar la asignación de recursos a un nodo preferencial:

```
crm configure location masterdrbd_in_cluster1
master_drbd_postgresql 100: 524743-db1
```

Resource agent de PostgreSQL.

El *resource agent* encargado de controlar el recurso de PostgreSQL (ocf:la_empresa:postgresql) es un *script* programado y personalizado de acuerdo a nuestro criterio, resultado de la experiencia que tuve con el *software* para arrancar, detener y validar si las instancias de PostgreSQL funcionan correctamente. Las siguientes líneas componen el *shell script* completo:

```
#!/bin/sh


# High-Availability PostgreSQL OCF resource agent for Linux
Debian® Squeeze.

# Description: Start, stop, check status and verify
environment of postgres daemon in a database server.

# Author:      Saúl Cortés Riquelme.

# License:     GNU General Public License (GPL).

# Derived in part from: PostgreSQL OCF resource agent of
Serge Dubrouski (sergeyfd@gmail.com).

# OCF parameters used:

# OCF_RESKEY_binfile          default: /etc/init.d/postgresql

# OCF_RESKEY_pidfile          default:
/var/run/postgresql/*.pid

# OCF_RESKEY_logfile          default:
/var/log/pacemaker/postgresql_ra

# OCF_RESKEY_pguser      default: postgres

# OCF_RESKEY_pghost      default: localhost

# OCF_RESKEY_pgport      default: 5432

# OCF_RESKEY_pgdb        default: postgres

# OCF_RESKEY_stop_time        default: 30

# OCF_RESKEY_sleep_time       default: 10

# Performance of this resource agent was tested in a Debian®
Squeeze O. S. with postgresql 8.4.17 version environment.
```

```
# To test in another version I recommend a previous
assessment of its operation.


## Initializing resource agent script.

. ${OCF_ROOT}/resource.d/heartbeat/.ocf-shellfuncs


## Exporting environment variables.

[ -n "$OCF_RESKEY_binfile" ] && export POSTGRES_BINFILE="-f
$OCF_RESKEY_binfile"

[ -n "$OCF_RESKEY_pidfile" ] && export POSTGRES_PIDFILE="-f
$OCF_RESKEY_pidfile"

[ -n "$OCF_RESKEY_logfile" ] && export POSTGRES_LOGFILE="-f
$OCF_RESKEY_logfile"

[ -n "$OCF_RESKEY_pguser" ] && export POSTGRES_USER="-f
$OCF_RESKEY_pguser"

[ -n "$OCF_RESKEY_pgport" ] && export POSTGRES_PORT="-f
$OCF_RESKEY_pgport"

[ -n "$OCF_RESKEY_pgdb" ] && export POSTGRES_DATABSE="-f
$OCF_RESKEY_pgdb"

[ -n "$OCF_RESKEY_stop_time" ] && export POSTGRES_STOPTIME="-
f $OCF_RESKEY_stop_time"

[ -n "$OCF_RESKEY_sleep_time" ] && export
POSTGRES_SLEEPTIME="-f $OCF_RESKEY_sleep_time"


## Reading action received.

action="$1"


## Declaring global variables.

binfile="$OCF_RESKEY_binfile"

pidfile="$OCF_RESKEY_pidfile"

logfile="$OCF_RESKEY_logfile"
```

```
pguser="$OCF_RESKEY_pguser"
pgport="$OCF_RESKEY_pgport"
pgdb="$OCF_RESKEY_pgdb"
stop_time="$OCF_RESKEY_stop_time"
sleep_time="$OCF_RESKEY_sleep_time"
psql_cmd='/usr/bin/psql'

## Assigning default values to unspecified variables.
if [ -z "$binfile" ]
     then
          binfile='/etc/init.d/postgresql'
fi

if [ -z "$pidfile" ]
     then
          pidfile='/var/run/postgresql/*.pid'
fi

if [ -z "$logfile" ]
     then
          logfile='/var/log/pacemaker/postgresql_ra'
fi

if [ -z "$pguser" ]
     then
          pguser='postgres'
fi
```

```
if [ -z "$pghost" ]
     then
          pghost='localhost'
fi

if [ -z "$pgport" ]
     then
          pgport='5432'
fi

if [ -z "$pgdb" ]
     then
          pgdb='postgres'
fi

if [ -z "$stop_time" ]
     then
          stop_time='30'
fi

if [ -z "$sleep_time" ]
     then
          sleep_time='10'
fi

## Show usage function.
```

```
usage()

{

     echo "\nUsage:\t$0 {start|stop|monitor|meta-
data|validate-all|usage}\n"

}


## Meta-data function (show XML meta-data).

postgres_metadata()

{

     cat <<END

<?xml version="1.0"?>

<!DOCTYPE resource-agent SYSTEM "ra-api-1.dtd">

<resource-agent name="postgresql">

<version>1.0</version>


<longdesc lang="en">

     Resource agent that controls postgres daemon in a
Debian® O.S. environment.

</longdesc>

<shortdesc lang="en">

     R. A. for PostgreSQL database server.

</shortdesc>


<parameters>

<parameter name="binfile" unique="0" required="0">

     <longdesc lang="en">

          Full path to script file that controls the
launching and halting of postgresql process.

     </longdesc>
```

```
     <shortdesc lang="en">postgresql binary file</shortdesc>

     <content type="string"
default="/etc/init.d/postgresql"/>

</parameter>


<parameter name="pidfile" required="0" unique="1">

     <longdesc lang="en">

          postgresql PID file that exist while postgres
daemon is running. This file never will exist if postgresql
is stopped. Please provide full path for this file.

     </longdesc>

     <shortdesc lang="en">

          Full path of postgresql PID file.

          </shortdesc>

     <content type="string"
default="/var/run/postgresql/*.pid"/>

</parameter>


<parameter name="logfile">

     <longdesc lang="en">

          The full path for PostgreSQL Resource Agent log
file, where principal events will be reported.

     </longdesc>

     <shortdesc lang="en">

          Full path of PostgreSQL Resource Agent log file.

     </shortdesc>

     <content type="string"
default="/var/log/pacemaker/postgresql_ra"/>

</parameter>
```

```
<parameter name="pguser">
     <longdesc lang="en">
          The user name who will manage postgresql process
and his databases.
     </longdesc>
     <shortdesc lang="en">
          User of postgresql.
     </shortdesc>
     <content type="string" default="postgres"/>
</parameter>


<parameter name="pghost">
     <longdesc lang="en">
          The host name or IP address to where postgresql
will be polling.
     </longdesc>
     <shortdesc lang="en">
          postgresql host.
     </shortdesc>
     <content type="string" default="localhost"/>
</parameter>


<parameter name="pgport">
     <longdesc lang="en">
          The Operative System port where postgresql is
listening.
     </longdesc>
     <shortdesc lang="en">
          postgresql port.
```

```
     </shortdesc>

     <content type="integer" default="5432"/>

</parameter>


<parameter name="pgdb">

     <longdesc lang="en">

          The database that will be used for monitoring the
postgresql response.

     </longdesc>

     <shortdesc lang="en">

          postgresql database.

     </shortdesc>

     <content type="string" default="postgres"/>

</parameter>


<parameter name="stop_time">

     <longdesc lang="en">

          The time to wait, in seconds, for postgresql full
stop before try to use 'kill' command.

     </longdesc>

     <shortdesc lang="en">

          Time for postgresql stopping.

     </shortdesc>

     <content type="integer" default="30"/>

</parameter>


<parameter name="sleep_time">

     <longdesc lang="en">
```

```
          The enough time, in seconds, to wait before
consider that postgresql is killed after such instruction.

     </longdesc>

     <shortdesc lang="en">

          Time to wait for kill postgresql.

     </shortdesc>

     <content type="integer" default="10"/>

</parameter>

</parameters>


<actions>

<action name="start"               timeout="120" />

<action name="stop"          timeout="120" />

<action name="monitor" depth="0"       timeout="30"
interval="30" />

<action name="meta-data"           timeout="5" />

<action name="validate-all"            timeout="5" />

<action name="usage"               timeout="5" />

</actions>


</resource-agent>

END

}


## Log function.

log_action()

{

     stage="$1"
```

```
        logtext="$2"
        now=`date +%F\ %T.%N`
        echo -e "$now\t$stage:\t$logtext" >> $logfile
}

## Kill function.
kill_postgres()
{
        log_action 'KILL' '\tKilling postgres...'
        kill -TERM $(pidof postgres) > /dev/null 2>&1
        sleep $sleep_time

        pidof postgres > /dev/null
        pidof_result="$?"

        if [ "$pidof_result" != "$OCF_SUCCESS" ]
             then
                  log_action 'KILL' '\tpostgres daemon killed by
TERM signal.'
                  return $OCF_SUCCESS
        fi

        kill -9 $(pidof postgres) > /dev/null 2>&1
        sleep $sleep_time

        pidof postgres > /dev/null
        pidof_result="$?"
```

```
     if [ "$pidof_result" != "$OCF_SUCCESS" ]
          then
               log_action 'KILL' '\tpostgres daemon killed by
KILL signal.'
               return $OCF_SUCCESS
     fi


     log_action 'KILL' '\tattempt to kill postgres was
failed.'
     return $OCF_ERR_UNIMPLEMENTED
}


## Start function.
postgres_start()
{
     log_action 'START' '\tStarting postgresql...'
     log_action 'START' '\tReceived parameters:'
          log_action '\t:' "\t\t\t\tbinfile     =  $binfile"
          log_action '\t:' "\t\t\t\tpidfile     =  $pidfile"
          log_action '\t:' "\t\t\t\tlogfile     =  $logfile"
          log_action '\t:' "\t\t\t\tpguser      =  $pguser"
          log_action '\t:' "\t\t\t\tpghost      =  $pghost"
          log_action '\t:' "\t\t\t\tpgport      =  $pgport"
          log_action '\t:' "\t\t\t\tpgdb        =  $pgdb"
          log_action '\t:' "\t\t\t\tstop_time   =  $stop_time
[s]"
          log_action '\t:' "\t\t\t\tsleep_time  =
$sleep_time [s]"
```

```
    rm -f $pidfile > /dev/null 2>&1
    log_action 'EXECUTION' "postgresql $action."

    $binfile start
    action_status="$?"

    if [ "$action_status" != "$OCF_SUCCESS" ]
        then
            log_action 'EXECUTION' "postgresql $action
action was failed!\t(Code: $action_status)"
            return $action_status
    fi

    while : ; do
        postgres_monitor
        monitor_status="$?"

        if [ "$monitor_status" = "$OCF_SUCCESS" ]
            then
                break;
        fi

        log_action 'START' '\tpostgresql still has not
started yet. Waiting...'
        sleep 1
    done

    log_action 'START' '\tpostgresql starts successful.\n'
```

```
     return $OCF_SUCCESS
}

## Stop function.
postgres_stop()
{
     log_action 'STOP' '\tStopping postgresql...'
     log_action 'EXECUTION' "postgresql $action."

     $binfile stop
     action_status="$?"

     if [ "$action_status" != "$OCF_SUCCESS" ]
          then
               log_action 'EXECUTION' "postgresql $action
action was failed!\t(Code: $action_status)"
               return $action_status
     fi

     count=0

     while [ $count -lt $stop_time ] ; do
          postgres_monitor
          postgres_status="$?"

          if [ "$postgres_status" = "$OCF_NOT_RUNNING" ]
               then
```

```
                    log_action 'STOP' '\tpostgresql stops
successful.'
                    rm -f $pidfile > /dev/null 2>&1
                    return $OCF_SUCCESS
          fi

          log_action 'STOP' '\tpostgresql still has not
stopped yet. Waiting...'

          count=`expr $count + 1`
          sleep 1
     done

     log_action 'STOP' "\tTimeout for stopping is finished
($stop_time [s]). Postgresql could not be stopped normally."
     kill_postgres
     kill_result="$?"

     if [ "$kill_result" != "$OCF_SUCCESS" ]
          then
               log_action 'STOP' "\tpostgres stop action was
failed!\t(Code: $kill_result)"
               return $OCF_ERR_CONFIGURED
     fi

     rm -f $pidfile > /dev/null 2>&1
     log_action 'STOP' '\tpostgres stops successful.'
     return $OCF_SUCCESS
}
```

```
## Monitor function.
postgres_monitor()
{
     log_action 'MONITOR' 'postgres monitoring...'

     ls $pidfile > /dev/null
     pidfile_status="$?"

     if [ "$pidfile_status" != "$OCF_SUCCESS" ]
          then
               log_action 'MONITOR' 'postgres PID file does
not exist!'
     fi

     pidof postgres > /dev/null
     pid_status="$?"

     if [ "$pid_status" != "$OCF_SUCCESS" ]
          then
               log_action 'MONITOR' 'postgres daemon not has
PID!'
     fi

     $psql_cmd -h $pghost -p $pgport -U $pguser $pgdb -c
"SELECT now();" > /dev/null 2>&1
     db_status="$?"

     if [ "$db_status" != "$OCF_SUCCESS" ]
```

```
          then

               log_action 'MONITOR' "Query to database $pgdb
has failed!\t(Code: $db_status)"


               if [ "$db_status" = '1' ]

                    then

                         log_action '\t:' '\t\t\t\tFatal
error (out of memory or file not found or something) occurred
while executing the psql command.'

               elif [ "$db_status" = '2' ]

                    then

                         log_action '\t:' '\t\t\t\tConnection
error (connection to the server went bad and the session was
not interactive) occurred while executing the psql command.'

               elif [ "$db_status" = '3' ]

                    then

                         log_action '\t:' '\t\t\t\tScript
error (the variable ON_ERROR_STOP was set) occurred while
executing the psql command.'

               fi

     fi



     if [ "$pidfile_status" = "$OCF_SUCCESS" ] && [
"$pid_status" = "$OCF_SUCCESS" ] && [ "$db_status" =
"$OCF_SUCCESS" ]

          then

               log_action 'MONITOR' 'postgres is running OK.'

               return $OCF_SUCCESS

     elif [ "$pidfile_status" != "$OCF_SUCCESS" ] && [
"$pid_status" != "$OCF_SUCCESS" ] && [ "$db_status" !=
"$OCF_SUCCESS" ]
```

```
          then
               log_action 'MONITOR' 'postgres is not
running.'
               return $OCF_NOT_RUNNING
          else
               log_action 'MONITOR' 'postgres is running
partially.'
               return $OCF_ERR_GENERIC
     fi
}

## Validate binaries function.
postgres_validate_all()
{
     if ! have_binary $SHELL
          then
               return $OCF_ERR_INSTALLED
     fi

     if ! have_binary $binfile
          then
               return $OCF_ERR_INSTALLED
     fi

     if [ ! -x "$binfile" ]
          then
               return $OCF_ERR_PERM
     fi
```

```
     if ! have_binary $psql_cmd
          then
               return $OCF_ERR_INSTALLED
     fi

     return $OCF_SUCCESS
}



## MAIN.

if [ "$#" != '1' ]
     then
          log_action 'USAGE' "\tThis Resource Agent only can
receive one argument.\t(args received: $#)\n"
          usage
          exit $OCF_ERR_ARGS
fi

case "$action" in
     meta-data)
          postgres_metadata
          exit $OCF_SUCCESS   ;;
     validate-all)
          postgres_validate_all
          exit $?             ;;
     usage)
```

```
          usage
          exit $OCF_SUCCESS   ;;
esac

postgres_validate_all
validate_status="$?"

if [ "$validate_status" != "$OCF_SUCCESS" ]
     then
          log_action 'VERIFY' "\tValidation function has
returned an error.\t(Code: $validate_status)\n"
          exit $validate_status
fi

case "$action" in
     start)
          postgres_start           ;;
     stop)
          postgres_stop            ;;
     monitor)
          postgres_monitor         ;;
     *)
          log_action 'ERROR' "\tThe action received ($action)
is not implemented on this Resource Agent script!\n"
          usage
          exit $OCF_ERR_UNIMPLEMENTED   ;;
esac
```