Índice

1. Introducción

El modelo sanitario tradicional ha sufrido importantes variaciones debido a cambios sociales, políticos y económicos. En la pasada década, los cambios demográficos, el incremento de los costes sanitarios, la necesidad de mejora de la calidad asistencial y la búsqueda de equidad social, hacen necesaria una nueva concepción de la asistencia sanitaria, en la cual las tecnologías de la información y la telecomunicación jueguen un papel importante. Así, mezcla del estado de madurez de las tecnologías médicas y de la *sociedad de la información* surge el concepto de telemedicina.

A partir del marco previamente expuesto, el GBT (Grupo de Bioingeniería y Telemedicina) de la Universidad Politécnica de Madrid y la ONG Ingeniería Sin Fronteras han iniciado investigaciones para el diseño de sistemas y servicios de comunicación adaptados a las necesidades del personal sanitario rural de los países de América Latina. El proyecto **Enlace Hispano Americano de Salud** (EHAS) [10] surge como una iniciativa integral que pretende transferir conocimientos tecnológicos y metodologías de acceso a información médica a cada uno de dichos países, favoreciendo que sean éstos quienes ofrezcan servicios específicos destinados a cubrir las necesidades de cada país.

1.1. Conceptos generales

Para comprender mejor el escenario sobre el cual va a funcionar nuestro sistema, deberemos describir en que tipo de establecimientos se ejecutará, y el esquema de la red que está en funcionamiento en la actualidad.

El sistema será usado en un entorno médico, concretamente en establecimientos de salud. Por un lado tenemos los **Centros de Salud**, de mayor jerarquía, situados en capitales de provincia o distrito, donde sí llega la línea telefónica. Un **Centro de Salud** es centro de referencia de varios **Puestos de Salud**. Dichos puestos están situados en poblaciones sin línea telefónica y mal dotadas en infraestructura de carreteras. La comunicación e intercambio de información entre éstos puede llevar horas e incluso días. La necesidad de comunicación es especialmente importante en estas zonas rurales en caso de un brote epidémico, desastre natural, reportes del sistema de información sanitaria y sistemas de recepción de medicamentos.

1.2. Topología de la red EHAS

Las investigaciones llevadas a cabo hasta ahora por el programa EHAS se centran en tres puntos fundamentales:

Radio VHF-UHF. La mayoría de Centros de Salud cuentan con conexión telefónica. No así los Puestos de Salud que dependen de ellos. Por esta razón, el esfuerzo investigador se está centrando en lograr el enlace de los Puestos de Salud con el teléfono más cercano. La opción de menor coste y mayor calidad la ofrecen los enlaces radio en las bandas VHF y UHF en condiciones estándar con anchos de banda de 12.5KHz. Estos enlaces permiten una velocidad de hasta 9.600 bps (bits por segundo), aunque en las subredes del proyecto EHAS actualmente en funcionamiento la velocidad de los enlaces es de 1.200 bps. El inconveniente de esta opción es la limitación de la longitud del enlace (aproximadamente 50 km) y la necesidad de contar con visión casi directa entre antenas.

Radio HF. En las situaciones de orografía complicada o larga distancia entre el Puesto de Salud remoto y el Centro de Salud, la opción idónea es un enlace HF directamente con la capital. Esta solución, más cara que la anterior, tiene sus limitaciones en la baja velocidad obtenida (300 bps).

Satélites de baja órbita (LEO). Sólo en casos excepcionales se contempla el empleo de satélites LEO para enlazar un Centro de Salud aislado con el resto de la red internacional EHAS. Esta solución permite 9600 bps de velocidad de transmisión y acceso a cualquier punto aislado, pero es la de mayor coste de infraestructura y explotación.

La topología de la que se parte contempla conexiones de radio, telefónicas y acceso a Internet. De forma genérica, como vemos en la figura 1, existen tres tipos de nodos:

Nodo Terminal. Es el emplazamiento del usuario final. Habitulamente se tratará de un Puesto de Salud aislado y normalmente sólo tiene un enlace digital vía radio con el Nodo Local (Centro de Salud). En ciertos casos puede tener un enlace HF directo a un Nodo Nacional o Internacional. El usuario utiliza programas convencionales de correo electrónico bajo Microsoft Windows.

Nodo Local. Es el lugar que centraliza la comunicación vía radio con varios Nodos Terminales que dependen de él. Gestiona las comunicaciones entre ellos sin coste de operación. El Nodo Local es un Centro de Salud ubicado en una localidad con línea telefónica, la cual utiliza, mediante conexiones periódicas, para comunicarse con el Nodo Nacional. El servidor utilizado es un sistema Linux con un servidor de correo que sirve como pasarela entre la red de radio y el nodo nacional.

Nodo Nacional. Centraliza la comunicación con todos los Nodos Locales. Además, es la pasarela de nuestra red con el resto de Internet.

Nodo Internacional. Situado en España, este nodo centraliza la información procedente de comunicación vía satélite para servir posteriormente de pasarela al resto de Internet.

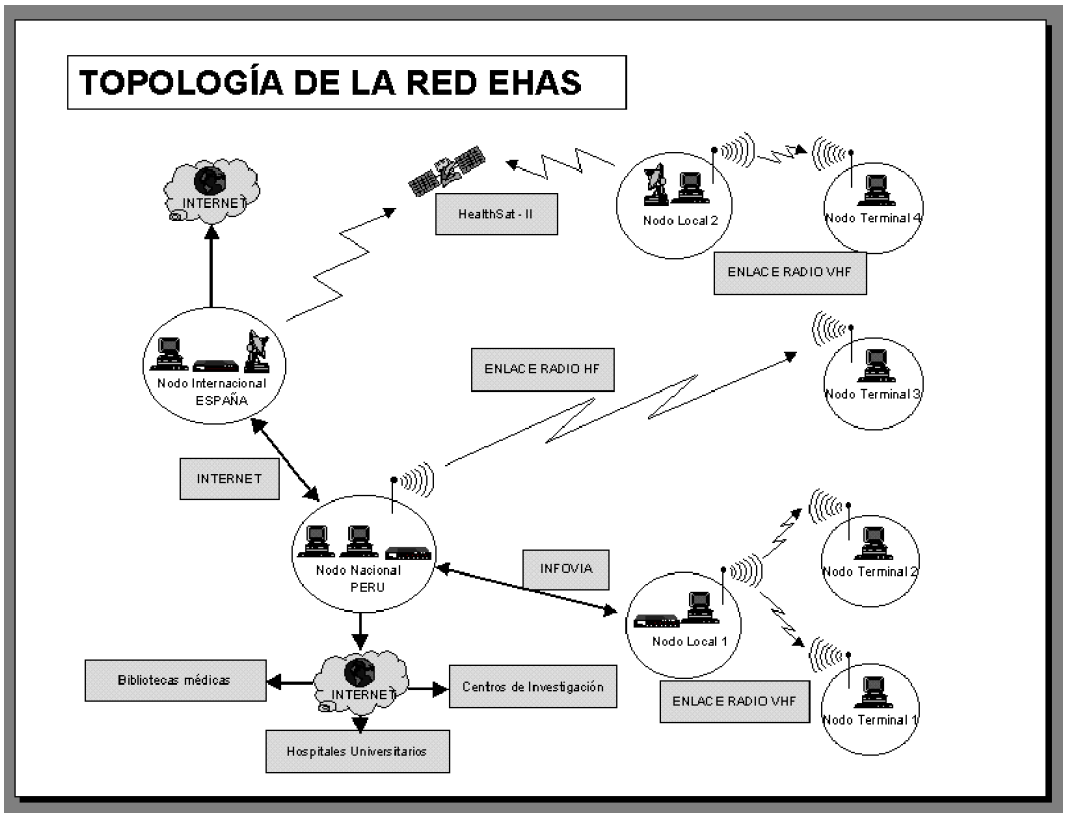


Figura 1: Topología de la red EHAS

El esquema anterior funciona en el caso que todos los Nodos Terminales tengan visión directa con su Nodo Local. Si para alguno de ellos resulta imposible conseguir un enlace en VHF con el Nodo Local, es necesario realizar un enlace directo con el nodo Nacional a través de HF (Onda Corta), con las ventajas y desventajas que antes mencionamos. Por último, en el caso extremo de tener un Nodo Local que no tenga acceso a línea telefónica, el correo se enruta a través del satélite LEO HealthSat II hasta el Nodo Internacional en España y de ahí se reenvía por Internet a su destino.

1.3. Objetivos

El presente proyecto se centra en la mejora de la red VHF-UHF. Para ello habrá que cubrir un cuadruple objetivo:

- Mejora del rendimiento de las subredes ya instaladas a 1200bps
- Estudio y diseño de toda la infraestrucura necesaria para la instalación de nuevas subredes a 9600bps con la mayor eficiencia posible.
- Mejora del coste y disminución del consumo de la infraestructura de comunicaciones.
- Posibilidad de administración remota de los subsistemas.

Para la consecución de estos objetivos se desarrollará un sistema que permita el uso de las aplicaciones estándar de correo electrónico (Netscape Messenger, Microsoft Outlook) permitiendo la recepción y envío de correo electrónico en clientes Windows que estén equipados con un módem y una radio como medio de comunicación.

Se propondrá, además, la compresión y cifrado de todos los datos que se transmitan en el enlace radio para mejorar la eficiencia y seguridad. El sistema permitirá, asimismo, la gestión remota de los clientes desde cualquier punto de Internet para solucionar posibles problemas de configuración o corrupción del sistema.

De igual forma, también será necesaria la prueba y evaluación de diferentes módems que permitan aumentar a 9600 bps la velocidad del sistema, actualmente a 1200 bps.

Otro aspecto a mejorar será el rendimiento de las aplicaciones estándar de correo electrónico, pues ofrecen rendimientos muy bajos debido a la incorrecta interacción entre el protocolo AX.25 [5] (usado en el enlace radio) y TCP/IP (usado por las aplicaciones de correo), para ello se hará un estudio de las aplicaciones más adecuadas para optimizar el enlace AX.25, para lo que se hará uso de aplicaciones SSH (*Secure Shell*) [23].

Por último se implementará el protocolo DAMA (*Demand Assigned Multiple Access*, Acceso múltiple asignado bajo demanda) [8]. Dicho protocolo permite a un ordenador ubicado en el centro geográfico, la asignación de turnos a los clientes que simultáneamente han establecido una conexión con el mismo. Con este protocolo se cubren los siguientes aspectos:

1. No será requisito imprescindible que todas las antenas que forman la red tengan visibilidad directa entre ellas como hasta ahora. Sólo se requerirá, como es lógico, que el cliente tenga visibilidad directa con la antena del servidor DAMA. Esto permite desplegar redes en zonas en las que hasta ahora se hacía imposible, o en las que se había hecho obligada la instalación de repetidores, con el consecuente encarecimiento del proyecto.

2. Por la razón antes mencionada, la potencia consumida por las antenas puede ser menor: el uso de antenas directivas y una longitud de alcance requerida menor así lo permiten. En un proyecto donde la energía se obtiene de paneles solares este aspecto es de gran importancia. De igual forma la altura de las antenas podrá ser menor, con un consiguiente ahorro en materiales y una instalación más sencilla.

3. La curva de carga pasa de ser la de un sistema CSMA (*Carrier Sense Multiple Access*) [20] a ser similar al de un sistema TDMA, con ligeras diferencias, por lo que lo denominaremos CSMA-DAMA. El elevado número de colisiones que hasta ahora se producían con el sistema CSMA se debe a que el tipo de radios usadas son *half-duplex* (no pueden trasmitir y recibir a la vez). Además, estas radios presentan tiempos de commutación elevados, por lo que entre que una radio ve el canal libre y decide transmitir hasta que efecivamente lo hace, pasa un tiempo significativo en el cual otra radio puede intentar otra trasmisión, produciéndose una colisión. Este problema queda resuelto con la implementación de DAMA.

2. Especificación de requisitos

El sistema hará uso de aplicaciones de correo electrónico ordinarias, siendo transparente para el usuario final la utilización del sistema radio. Además, como en cualquier desarrollo de un sistema, el análisis de requisitos debe enfocarse con el objetivo que la configuración e instalación de nuestra aplicación sea lo más sencilla posible. Para ello debemos crear un entorno amigable y fácil de usar.

Los requisitos que pediremos al sistema y a las aplicaciones que lo integran serán:

Facilidad de instalación, mantenimiento y configuración. Tanto la instalación como la configuración deben ser de fácil comprensión, reduciéndose al mínimo necesario para un funcionamiento eficaz de las aplicaciones.

Robustez. El sistema debe ser lo suficientemente resistente a una utilización errónea por parte del usuario. También deberá ser capaz de recuperarse ante cualquier fallo interno del sistema o del enlace radio. Asimismo la aplicación debería automantenerse sin intervención del usuario, o sea, no debería deteriorarse ni corromperse con el paso del tiempo.

Flexibilidad en los requisitos hardware. La aplicación debe adaptarse a diversas configuraciones hardware, de manera que funcione con el mayor numero posible de ellas.

Seguridad. La aplicación debe ser capaz de ofrecer un entorno mínimo de seguridad, de manera que el usuario demuestre su identidad de forma inequívoca, aunque la autentificación se deberá realizar de forma ágil y cómoda. El sistema deberá asegurar que la información transferida estará protegida de uso no autorizado.

Transmisión de correo electrónico. Este es el requisito clave: el usuario desea trasmitir correo electrónico desde su ubicación al resto del mundo. Este debe realizarse de forma trasparente al usuario.

Alta compatibilidad. Interoperabilidad con los protocolos de correo estándar empleados en la actualidad en Internet (SMTP y POP3). Su utilización en nuestro sistema asegura que será altamente compatible.

Requisitos software. Los requisitos software dependerán en función del lado cliente o el lado servidor. El cliente deberá funcionar bajo los sistemas operativos Windows 95/98. El servidor deberá trabajar bajo GNU/Linux.

Requisitos hardware. Los requisitos hardware mínimos vendrán impuestos por el sistema operativo que deba tener el ordenador. En el caso del lado cliente (Windows) el ordenador mínimo deberá ser algo más potente que el que necesitaremos para el lado del cliente (Linux).

Flexibilidad geográfica de los Puestos de Salud. Los Puestos de Salud ya no deberán tener visión directa entre ellos, únicamente será necesaria con el servidor central.

Aumento de velocidad. En la red ya instalada a 1200bps se requiere una mayor eficiencia que aumente la velocidad del sistema. De igual forma se pedirá esta eficiencia para las nuevas subredes de 9600bps.

Estudio de módems. Se hará la evaluación de varios módems de bajo coste necesarios para las nuevas redes a 9600bps.

Ahorro en energía e infraestructura. El sistema deberá permitir la instalación de antenas directivas en los clientes, lo que redunda en un gasto menor de energía y una altura de torres inferior.

Gestión remota. Se permitirá acceder a cualquier punto de las subredes (incluidos los clientes Windows) desde cualquier ordenador conectado a Internet. La gestión remota permitirá algún mecanismo de restauración automática de imágenes de seguridad (*backups*).

3. Diseño

3.1. Uso de aplicaciones TCP/IP

3.1.1. Arquitectura de protocolos

Dado que en la red se van a utilizar protocolos TCP/IP, es preferible describir la arquitectura de protocolos desde el punto de vista de esta arquitectura, en lugar de la tradicional pila OSI (Open Systems Interconnection). El conjunto de protocolos TCP/IP puede dividirse en cinco capas específicas:

Capa de aplicación Proporciona una comunicación entre procesos o aplicaciones en computadores distintos.

Capa de transporte Proporciona un servicio de transferencia de datos extremo a extremo entre aplicaciones. Oculta los detalles de la red o subredes, subyacente a la capa de aplicación.

Capa de red, o internet Está relacionada con el encaminamiento de los datos del computador origen al destino a través de una o más redes conectadas por dispositivos de encaminamiento.

Capa de enlace Relacionada con la interfaz lógica entre sistema final y una subred.

Capa física Define las características del medio de transmisión, la tasa de señalización y el esquema de codificación de las señales.

NIVEL APLICACIÓN	TELNET, FTP, etc
NIVEL TRANSPORTE	TCP/UDP
NIVEL RED	IP
NIVEL ENLACE	AX.25
NIVEL FÍSICO	RADIO

3.1.2. Encapsulación TCP/IP

Una de las opciones clásicas para permitir el uso de TCP/IP en entorno radio es la encapsulación de IP en los paquetes AX.25 (tal como se ve en la pila de protocolos del apartado anterior). En la red que está ya montada se optó por el programa **Ethrax25** [11], que realiza dicha encapsulación. Con esta configuración se podía acceder al servidor de la forma habitual. En las pruebas realizadas (se hará una comparativa entre prestaciones en el capítulo 10) se vió que el rendimiento de Ethrax25 era muy bajo, aproximadamente de 400 bps (en un enlace de 1200 bps).

El problema que encontrábamos con Ethrax25 es que sólo era capaz de establecer conexiones en **modo no conectado** (ver Anexo B), modo en el que la eficiencia del enlace AX.25 no se aprovecha al máximo. Además, era el protocolo TCP el encargado de pedir retransmisiones en caso de errores, y no siendo TCP un protocolo adaptado a un medio *half duplex* con altos tiempos de commutación como el de un enlace radio, el rendimiento del sistema era bajo.

En este punto se realizaron pruebas de encapsulación TCP/IP sobre AX.25 en **modo conectado** comunicando dos estaciones Linux. El rendimiento que se observó era nuevamente muy bajo aunque por causas distintas al caso anterior. El principal problema era la incorrecta interacción entre el protoco TCP y AX.25: en el caso de que se produjeran errores o colisiones, se enviaban peticiones de retransmisión desde el nivel TCP y también desde el nivel AX.25, pues ambos protocolo implementan funciones de control de errores, reordenado y petición de retransmisiones. Esta duplicidad de funciones relentizaba la comunicación y los temporizadores de espera del nivel TCP iban aumentando gradualmente hasta llegar en la mayoría de casos a la desconexión por inactividad.

Con tal de aislar si el problema estaba realmente en la interacción entre TCP y AX.25, se hicieron diversas pruebas consistentes en la transmisión de datos AX.25 puros (sin encapsulación) y por tanto sin la intervención del protocolo TCP. Se vio que en los mismos enlaces donde antes se conseguían velocidades muy bajas con la encapsulación IP, ahora se obtenían tasas de transmisión muy altas, al límite de la capacidad de un canal *half-duplex*:

- 5500 bps de datos reales en enlaces de 9600bps.
- 1066 bps de datos reales en enlaces de 1200bps.

Por ello se decidió que los datos a enviar por el canal radio sería de paquetes AX.25 sin encapsulación de ningún tipo. Pero dado que el sistema tiene que dar servicio a más de un puerto destino (como mínimo dos: SMTP y POP3), se debía implementar algún sistema que permitiera multiplexación de puertos, para acceder a los diferentes servicios. La opción escogida fue el uso de Secure Shell (SSH) (ver Anexo E).

Secure Shell es un protocolo definido en el RFC de T.Ylonen [23] con aplicaciones disponibles que lo implementan en plataformas Windows y Linux, formada por una parte cliente y otra parte servidor. En nuestro sistema necesitaremos un cliente para Linux y Windows, y un servidor para el servidor Linux, que permita la entrada remota desde los clientes. El sistema SSH tiene las siguientes prestaciones:

Métodos de autenticación variados. El usuario se puede identificar mediante una contraseña o por un sistema automático de autenticación basado en RSA.

Nivel de compresión variable. Permite la elección de nivel de compresión de los datos, lo que resultará en una transmisión efectiva de datos a mayor velocidad.

Multiplexación de puertos. Permite que distintos puertos de la máquina local estén redireccionados a través del túnel seguro SSH que se ha creado. Con esta característica nos es posible acceder a diferentes servicios (POP3, SMTP) en la máquina servidora.

Nuestra tarea es comunicar los clientes con el servidor mediante el enlace radio, que como ya hemos dicho, transmitirá datos AX.25 sin encapsulación IP. Para ello se ha programado unos programas de escucha (a partir de ahora lo llamaremos *daemon*), que sirvan de pasarela entre los datos del enlace radio y los de los clientes y servidores SSH. El esquema propuesto es el reflejado en la figura 2. En los próximos capítulos se detalla el funcionamiento de los *daemons* tanto para el cliente (Windows) como el servidor (Linux).

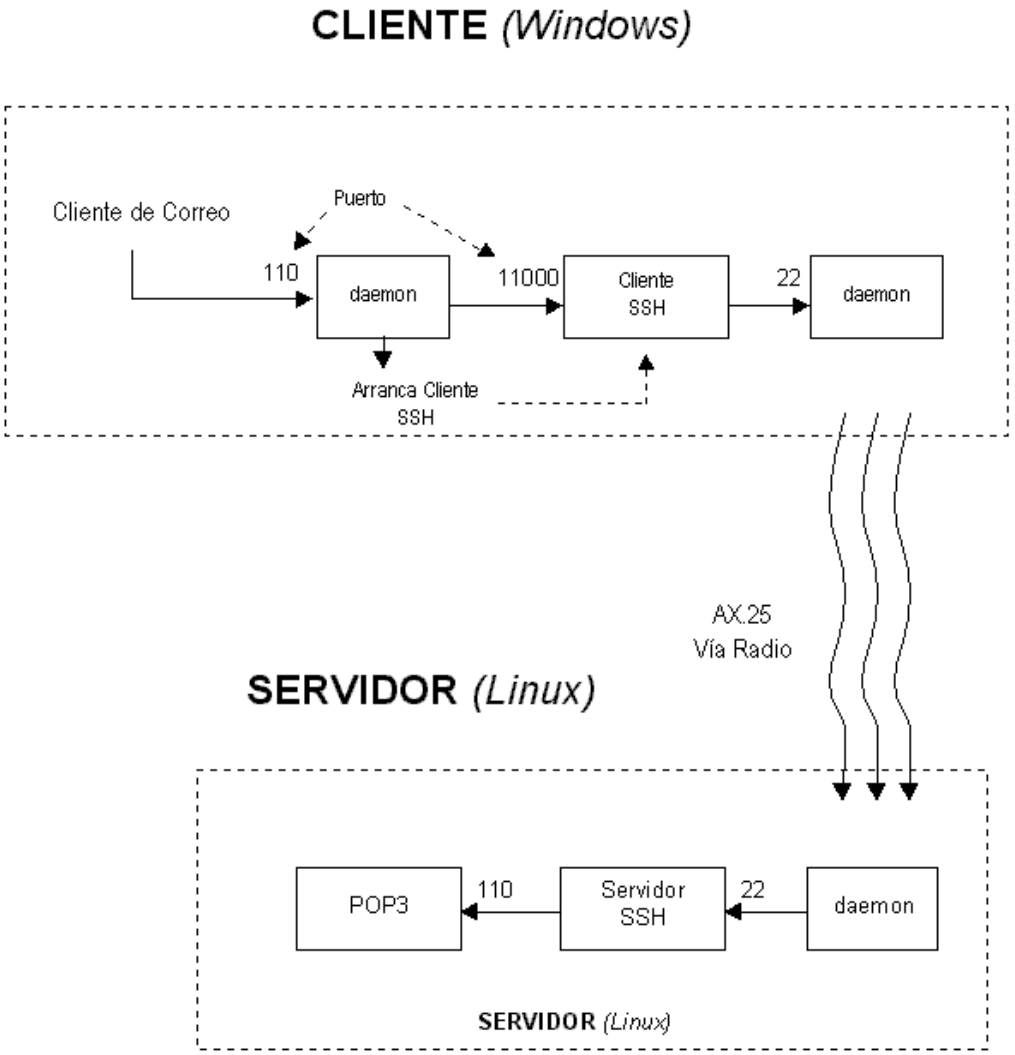


Figura 2: Esquema general de los *daemons*

3.2. DAMA

Aparte de permitir la transmisión de paquetes TCP/IP en el enlace radio, uno de los objetivos principales del presente proyecto es la implementación del protocolo DAMA.

Este protocolo se encarga de asignar turnos a los diferentes clientes que estén siendo servidos de forma simultánea, de forma que se reduzcan las colisiones de paquetes al mínimo.

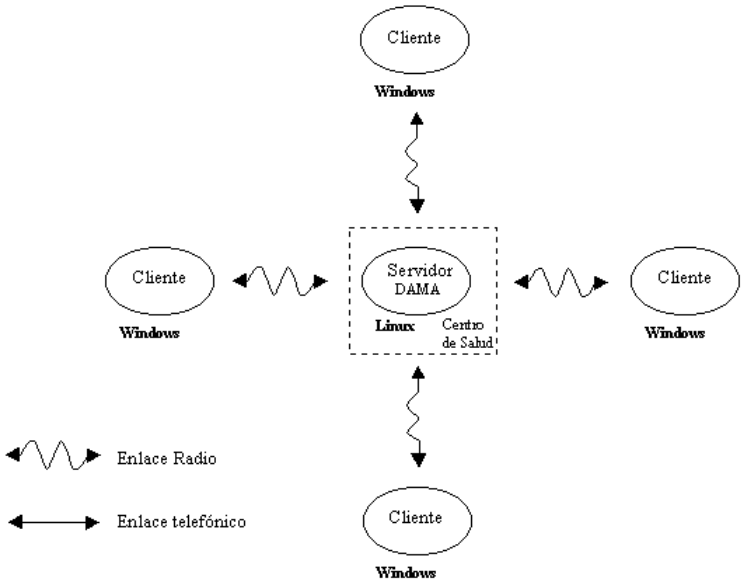


Figura 3: Servidor DAMA en el Centro de Salud

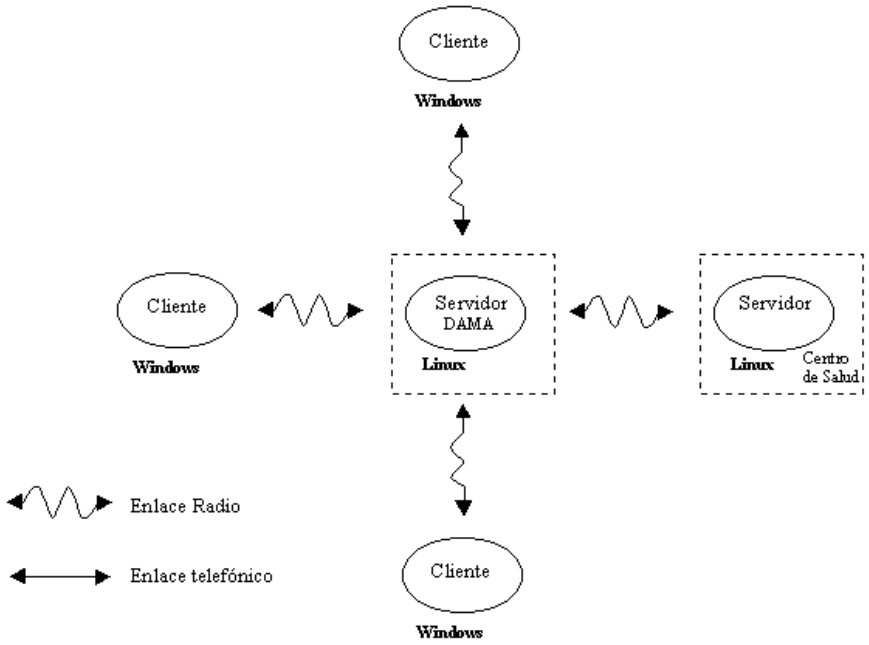


Figura 4: Servidor DAMA dedicado

La implementación de DAMA plantea dos nuevos escenarios que en la configuración actual no existían. Dado que el ordenador que haga las funciones de ordenador central DAMA, al que denominaremos *DAMA-Server* (Servidor DAMA) ó *DAMA-Master* (Maestro DAMA), debe estar en el centro geográfico de la subred, puede suceder que en ese punto esté el Centro de Salud (figura 3) o que no lo esté (figura 4). Es habitual que el Centro de Salud esté en el centro por razones topográficas, pero no siempre es así, por lo que deberemos tener en cuenta las dos alternativas y desarrollar las aplicaciones necesarias que cubran ambas posibilidades.

4. Estudio de aplicaciones auxiliares

Para el funcionamiento del sistema que constituye este proyecto se usaron, aparte de aplicaciones propias que se verán más adelante, aplicaciones ya existentes. Aquí veremos la funcionalidad que tienen, como se usan y el porqué de su elección.

4.1. Cliente

4.1.1. Implementación de AX.25

Como software encargado de implementar el protocolo AX.25 se han estudiado dos alternativas: Flexnet y AgwPacketEngine (AGW a partir de ahora).

Flexnet Flexnet [12] es un programa desarrollado por Thomas Sailer en 1995. Las primeras versiones que se hicieron de este programa funcionaban solamente en MsDos, aunque posteriormente se añadieron módulos adicionales que permitían su funcionamiento en Windows.

Aunque las pruebas que se han hecho con él con tarjetas de sonido han sido satisfactorias a 1200bps (ver apartado 5.1), ha sido desestimado por varias razones:

- Tiene un número de módulos reducidos para el uso de diferentes módems. Por ejemplo es incapaz de controlar módems muy extendidos en la actualidad como TNCs o PicPar (ver Capítulo 5).
- El interfaz para comunicarse con él es complejo y poco versátil.
- Se trata de un programa en origen pensado para MsDos, con lo que es menos compatible y difícil de configurar que un programa programado especialmente diseñado para Windows.

AGW AGW [1], aplicación desarollada por George Rossopulos, es mucho más reciente que Flexnet (empezó a ser distribuido en 1999), por lo que está totalmente programado para Windows 95/98/2000. Además, se trata de un programa de uso libre y en constante actualización.

AGW ha sido el programa elegido para implementar el protocolo AX.25 por estas razones:

- Extensa biblioteca de más 100 módems utilizables (entre ellos, todos los que se han probado en el proyecto)
- El interfaz para comunicarse con él es simple y versátil. Usa un puerto TCP local del ordenador para intercambiar datos con el programa usuario. Dicho interfaz es potente y está totalmente documentado (ver Anexo F)

4.1.2. Secure Shell

Según se ha visto en el capítulo de diseño, necesitaremos para la realización del proyecto un cliente de Secure Shell para Windows. Este cliente deberá disponer de las siguientes prestaciones:

- Redirección de puertos locales.
- Autenticación mediante sistema de claves públicas RSA.
- Compresión y cifrado de los datos.
- Fácil interacción con control no manual, es decir, sin intervención del usuario final.
- Fácil configuración.

El programa escogido, y que cumple las especificaciones requeridas, es el SSH32 v1.0 [19] desarrollado en 1999 por Cedomir Igaly. Además se trata de un programa *freeware* (gratuito), aunque no se ofrece el código fuente.

4.2. Servidor

4.2.1. Implementación de AX.25

En Linux la implementación de AX.25 ya viene en el kernel [16]. De tal forma que sólo hay que habilitar en la configuración del kernel el protocolo AX.25 y los módulos de los módems que se vayan a usar. Cuando se implemente DAMA, habrá que modificar el kernel y recompilarlo, pero no necesitaremos para ello ninguna herramienta que no venga por defecto en un sistema Linux.

4.2.2. Secure Shell

El Secure Shell, tanto el cliente como el servidor, vienen en cualquier distribución libre de Linux. Como es habitual en Linux, la implementación de SSH existente es extremadamente versátil y cubre todas las especificaciones pedidas en el apartado anterior.

5. Estudio de módems

En las subredes ya existentes en el programa EHAS se instalaron TNCs Plus a 1200bps. Como en las subredes que se vayan a instalar en el futuro se ha previsto aumentar la velocidad hasta 9600bps, se hace necesario el estudio de nuevos dispositivos hardware que lo permitan.

En este capítulo analizaremos las prestaciones de varios módems y veremos cual es el más adecuado para el proyecto. Este es un aspecto importante, pues como hemos ido viendo a lo largo del desarrollo del proyecto, no todos ellos dan las prestaciones deseadas.

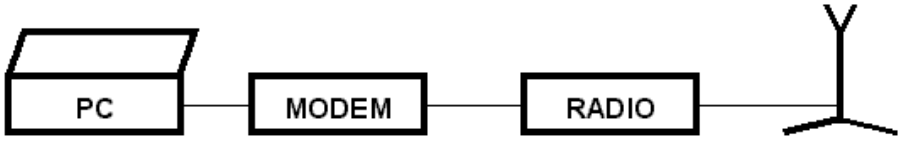


Figura 5: Esquema de dispositivos de comunicación

Los módems que se van a analizar, excepto las tarjetas de sonido cuyo uso es más general, están diseñados específicamente para radios VHF/UHF con canales estándar de 12.5 KHz. Las radios que se usan en este proyecto son *half-duplex*, esto es, en un momento determinado sólo pueden transmistir o recibir. Cada cambio de modo transmisión a recepción o viceversa requiere un tiempo mínimo anterior y posterior a la transmisión. Dichos tiempos reciben el nombre de *txdelay* y *txtail* (en el apartado 6.5.1 se estudia con más detalle).

Para el entorno Windows el programa elegido para hacer las pruebas ha sido el AGW. En las pruebas con tarjetas de sonido inicialmente también se hicieron pruebas con Flexnet, aunque al final se desechó por las razones expuestas en el capítulo anterior. En el entorno Linux se hace uso de los módulos que vienen en el kernel.

5.1. Tarjetas de sonido

La alternativa de usar tarjetas de sonidos es muy atractiva. Dado que es un dispositivo que viene instalado en cualquier portátil, el coste adicional sería nulo. Además permite una gran versatilidad, pues el cambio de velocidad de transmisión, algoritmos de recepción, modulaciones, etc, se traduce en un simple cambio en el software, haciendo su actualización sencilla, rápida y barata. El único problema que presentaban las tarjetas de sonido *a priori* era una mayor carga de la CPU, pues debe hacer toda la tarea de modulación y demodulación. El único elemento *hardware* adicional era el montaje de una pequeña placa para enviar la petición de transmisión a la radio, mediante la línea PTT (*Push To Talk*) cuyo esquema eléctrico se puede ver en la figura 6.

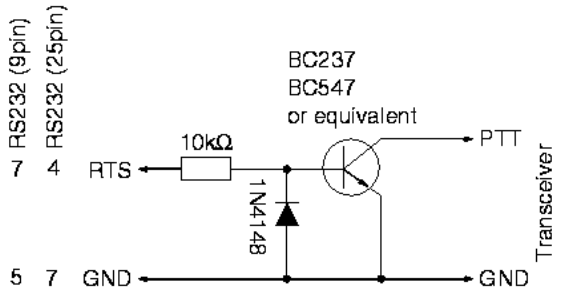


Figura 6: Esquema de generación de PTT en tarjeta de sonido

5.1.1. Pruebas

Todas las pruebas realizadas a 1200 bps, con independencia del tipo de tarjeta de sonido, daban resultados satisfactorios en ordenadores con Linux. En cambio, en Windows sólo Flexnet era capaz de transmitir y recibir correctamente a esa velocidad, no así AGW. Al ser AGW un programa que usa la API (*Application Programming Interface*, Interfaz de programación de aplicaciones) de sonido de Windows, que tienen una gran latencia, la velocidad no era suficiente para la modulación y demodulación en el portátil de pruebas, equipado con un Pentium 133Mhz.

Además, las pruebas realizadas con varios ordenadores y tarjetas de sonido de diferentes marcas a 9600bps ha ofrecido resultados muy dispares. Sólo ordenadores suficientemente potentes (Pentium 150) equipados con tarjetas de sonido de prestaciones muy elevadas (SB AWE 32, no disponibles en portátiles), han sido capaces de transmitir a esa velocidad.

Por lo que se ha visto, el hecho que las tarjetas de sonido no estén especificamente diseñadas como modulador/demodulador de datos sino como dispositivo de entrada/salida de audio de cálidad media, es decisivo en un comportamiento tan poco estable. Especialmente se ha detectado como un factor decisivo en el impacto en el rendimiento el tipo de filtros que tienen las tarjetas de sonido a la entrada y a la salida de audio. La función de dichos filtros es adaptar niveles de sonido, impedancias y respuesta frecuencial para micrófonos (entrada) y altavoces (salida), lo que en ningún caso es idóneo para la transmisión fidedigna de señal digital modulada.

5.1.2. Conclusión

Dado que ninguna tarjeta de sonido de un portátil, con independencia del sistema operativo, ha sido capaz de transmitir a 9600 bps, la opción de las tarjetas de sonido queda desechada.

5.2. TNC

Figura 7: TNC2 Plus

Las TNCs son dispositivos especializados para radiopaquete (*packet radio*), se comunican con el ordenador por el puerto serie y tienen la capacidad de realizar por si mismas las funciones de acceso al medio. Para ello, es capaz de almacenar los paquetes que se transmiten, provocando una menor carga del ordenador, y de almacenar los paquetes recibidos para ir sirviéndolos posteriormente.

En las subred en funcionamiento se estaban usando TNCs a 1200bps, con lo cual se tenía una experiencia sobre el funcionamiento de las mismas. Las principales desventajas que se habían encontrado es que a veces se producían paradas prolongadas sin explicación alguna e incluso algunas quedan totalmente inoperativas por salir de modo KISS, único modo en el cual el ordenador es capaz de comunicarse con la TNC y transmitir datos.

5.2.1. Pruebas

En las pruebas se han usado el modelo TNC2 Plus (compatibles con el estándar de la TAPR [21]) fabricadas por el grupo de radioaficionados Digigrup [9]. A pesar de las dificultades anteriormente descritas se han realizado pruebas con TNCs 1200 y 9600 bps (éstas equipadas con módems compatibles G3RUH [13]). Una de las primeras cosas que se vio era que, efectivamente, las TNCs tienen un comportamiento anómalo y después de un tiempo de inactividad los parámetros que controlan el acceso al medio (p persistencia, slottime, etc) se perdían y se volvían a establecer los parámetros por defecto con lo que la comunicación se relentizaba considerablemente. Por otra parte, y también sin razón aparente, la TNC salían de modo KISS y se hacía obligado ponerla de nuevo en ese modo para trabajar con ella.

Aparte de estos problemas, en las pruebas de prestaciones a 1200 bps, la TNC funciona correctamente, obteniendo las prestaciones siguientes:

Velocidad del enlace (bps)	1200
Num Paq enviados	1000
Num Paq retransmitidos	0
Velocidad efectiva (bps)	1066

Figura 8: Prestación de las TNCs a 1200bps

A 9600 bps, en cambio, la TNC presentaba algunos errores en la transmisión (aparición de REJs (rejects, ver Anexo B) según se ve en la éste cuadro:

Velocidad del enlace (bps)	9600
Num Paq enviados	1000
Num Paq retransmitidos	60
Velocidad efectiva (bps)	5586 bps

Figura 9: Prestación de las TNCs a 9600bps

5.2.2. Conclusión

Con los resultados obtenidos en las pruebas, la opción de usar TNCs como módems ha sido desestimada por varias razones:

- El comportamiento inesperado de las TNC, pérdidas de los parámetros de acceso al medio, *resets* no deseados, las hacen una opción poco recomendable en entornos donde la intervención exterior debería ser mínima, pues obliga al desplazamiento de personal especializado a zonas de difícil acceso.
- Las TNCs no son el *hardware* adecuado para la implementación de DAMA. Cuando un cliente recibe la orden de no enviar más paquetes de información, puede ser que éstos ya estén en la TNC almacenados para ser transmitidos, por lo que no hay forma de evitar el envío y entorpecerán de forma inevitable la comunicación de otro cliente.

5.3. YAM

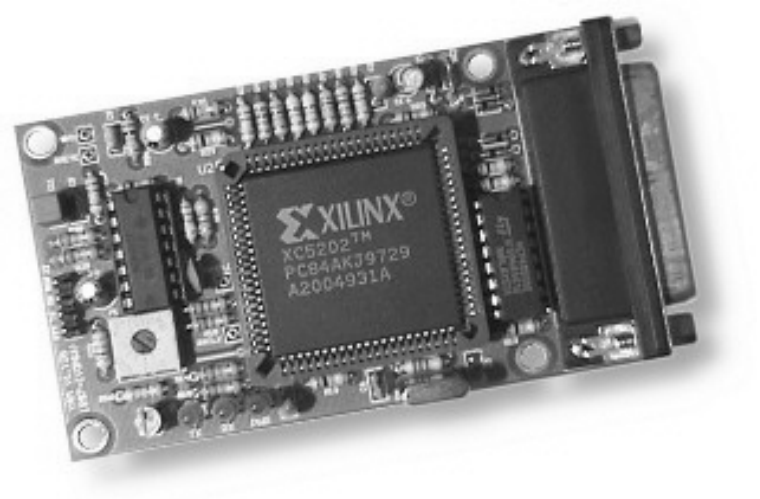


Figura 10: Módem Yam

Los YAM son módems usados en Packet Radio desarrollados por Nico Palermo en 1997 [22]. Se conectan al ordenador mediante el puerto serie y permiten una velocidad de transmisión de 9600bps siguiendo el estándar G3RUH. A diferencia de la TNC no hace funciones de almacenamiento de paquetes, así que los octetos que envía o recibe los trata de uno en uno.

5.3.1. Pruebas

Las pruebas con YAM en Linux dieron resultados bastante satisfactorios. En cambio, cuando se probaron en los clientes Windows, el módem sólo era capaz de recibir correctamente. En el momento que tenía que transmitir, el otro interlocutor era incapaz de recibir más de un 10 % de los paquetes que el YAM había enviado.

5.3.2. Conclusión

Debido a su rendimiento tan pobre en el sistema operativo Windows, al menos con el programa AGW, el módem YAM ha sido desestimado. Además, los módems YAM presentan el problema que se programan cada vez que se conectan al puerto serie por lo que una mínima pérdida de alimentación lo inutiliza hasta que volviera a ser programado.

5.4. PicPar

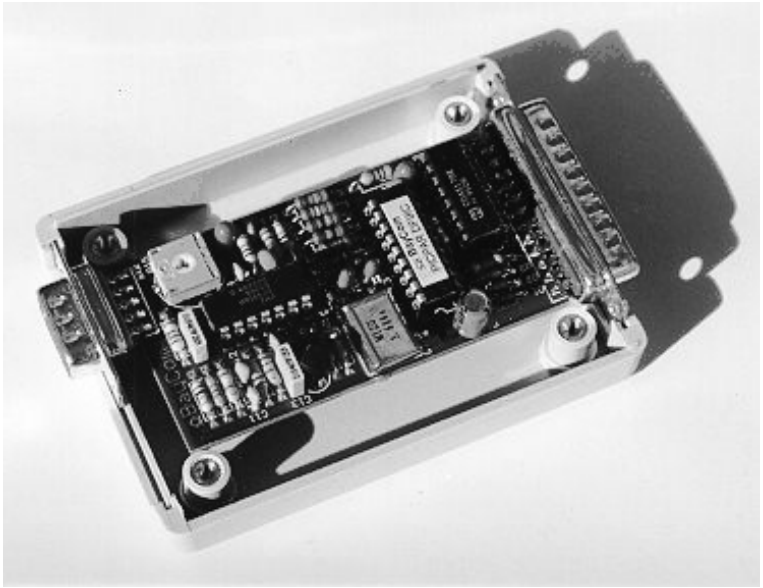


Figura 11: Modem PicPar

El módem Pipar es muy similar al YAM con la diferencia que se conecta al puerto paralelo en vez de al serie. Este módem fue diseñado ha sido diseñado por el grupo de radioaficionados alemán Baycom [4].

Velocidad del enlace (bps)	9600
Num Paq enviados	1000
Num Paq retransmitidos	100
Velocidad efectiva (bps)	5512 bps

Figura 12: Prestación de los PicPar a 9600bps

5.4.1. Pruebas

Las pruebas con PicPar han sido bastante satisfactorias. Su funcionamiento era bastante estable, aunque al igual que en las TNC, seguían presentando algunos REJs en la comunicación. Se obtuvo la tabla de prestaciones de la figura 12.

5.4.2. Conclusión

Aunque el número de retransmisiones haya sido mayor que el de una TNC, su funcionamiento ha sido el más estable de todos los módems usados. Dado que el programa que controla su funcionamiento ya está en el microcontrolador PIC que usa, es capaz de seguir funcionando a pesar de que durante algún momento pierda la alimentación, lo que es positivo en entornos aislados. Esta es una característica muy atractiva que no encontramos ni en TNCs ni en el módem YAM.

Además, es un módem de precio asequible y que se puede encontrar con facilidad en las tiendas distribuidoras de material para radioaficionados. En caso de que se dejara de fabricar, los esquemas están disponibles en la página web de varios grupos de radioaficionados.

6. Implementación en el cliente

6.1. Introduccion

En una red normal, las aplicaciones de correo electrónico se conectan a los puerto de SMTP o POP3 del servidor. En nuestro caso la conexión directa a través de red TCP/IP con el servidor no es posible, por lo que habrá que realizar un túnel que haga de intermediario entre el cliente de correo y el servidor.

6.2. Protocolo de actuación del *daemon* cliente

El *daemon* para el cliente Windows ha sido programado en Microsoft Visual C 5.0. Su función es la de escuchar los puertos SMTP y POP3, lanzar el cliente SSH cuando se produzca la conexión y servir de pasarela entre los datos del socket SSH y los datos del enlace radio (enlace AX.25). El listado completo del programa se puede encontrar en el disco que acompaña el proyecto (`daemon.cpp`). El protocolo de funcionamiento del *daemon* es el siguiente:

1. El *daemon* está a la espera de conexiones del cliente de correo a los puerto SMTP y POP3 locales.
2. Cuando se produce la conexión por parte del cliente de correo, se mira si ya hay una conexión activa en el otro puerto. Si aún no hay una conexión abierta, se invoca al cliente SSH.
3. El cliente SSH está configurado para conectarse al puerto local SSH (por defecto 22). El *daemon* nuevamente intercepta esta conexión y usa el programa AGW para crear el enlace AX.25 con el servidor. Para ello envía la petición de conexión AX.25.
4. Una vez el servidor contesta a nuestra petición de conexión empieza el intercambio de información entre cliente y servidor. En esta primera etapa se lleva a cabo la autentificación.
5. En el momento en que la autentificación és valida, el cliente SSH, previamente configurado, abre unos puertos en la máquina local. Dichos puertos están redirigidos de tal forma que todo lo que se escribe en ellos va al puerto adecuado en el servidor, a través del túnel SSH. A los puertos redirigidos (que son dos, uno para SMTP y el otro POP3), les llamaremos puertos SMTPbis y POP3bis, respectivamente.

6. En este momento el *daemon* hace función de doble pasarela:
   - Entre el puerto SMTP/POP3 y el puerto SMTPbis/POP3bis.
   - Entre el puerto SSH y el enlace radio AX.25
7. Si en este momento se pide una nueva conexión en el otro puerto libre (POP3 ó SMTP), ya no es necesario crear el túnel SSH ni la conexión AX.25, pues ya han sido creados. Así que el *daemon* sólo hará la función de pasarela entre el puerto STMP/POP3 y el correspondiente puerto SMTPbis/POP3bis.
8. Cuando se cierran todos los puertos SMTP/POP3 abiertos, el *daemon* cierra el socket con el cliente SSH y hace que el AGW mande una petición de desconexión AX.25, para cerrar el enlace AX.25 que ya no se necesita.

6.3. Estructura del *daemon* cliente

El *daemon* se divide en tres partes diferenciadas:

main: En este hebra (*thread*) se controlan las peticiones de conexión del cliente SSH local y la conexión a nivel AX.25 con el servidor. El pseudocódigo de esta función es el descrito a continuación. Particularidades del protocolo AX.25, como el uso de *callsings* (indicativos), se amplía en el Apéndice B:

```
arranca AGW;
creamos socket SSH;
bind socket SSH;
listen socket SSH;
crea Hebra de Escucha del puerto SMTP;
crea Hebra de Escucha del puerto POP3;
registra IndicativoLocal;
bucle infinito
        si recibida llamada AX.25
                arranca Shell;
        si recibida petición del Socket SSH
                mandamos petición de conexión AX.25;
                si conexión no lograda
                        volver inicio bucle infinito;
                crea Hebra obtener_hora para la sincronización de relojes;
                mientras Socket SSH y Socket AX.25 abiertos
```

```
                    esperar hasta que haya datos para leer;
                    si datos en socket SSH
                        leer socket SSH;
                        escribir socket AX25;
                    si datos en socket AX25
                        leer socket AX25;
                        escribir socket SSH;
                si socket SSH no cerrado
                    cerrar SocketSSH;
                desconexión AX25;
    fin bucle infinito
```

thread_tcp: Como se ha visto, la hebra *main* lanza a su vez dos *hebras* de tipo *thread_tcp*, una para el puerto SMTP y el otra para POP3. Se encargan de escuchar en esos puertos y de arrancar el cliente SSH cuando sea necesario. Sse ha tomado como ejemplo SMTP, idéntico para POP3:

```
creamos socket SMTP;
bind socket SMTP;
listen socket SMTP;
bucle infinito
    si recibida petición puerto SMTP
        si no existe túnel SSH
            ejecutamos cliente SSH32;
        connect socket SMTPbis;
        si conexión no lograda
            volver principio bucle infinito
        mientras socket SMTP y SMTPbis abiertos
            esperar datos de socket SMTP y STMPbis;
            si Datos en socket SMTP
                leer socket SMTP;
                escribir socket SMTPbis;
            si Datos en socket SMTPbis
                leer socket SMTPbis;
                escribir socket SMTP;
        Si no existe conexión POP3
            cerrar(Socket SSH);
fin bucle infinito
```

shell: Si se recibe una petición de administración remota este función se encarga de ejecutar el intérprete de órdenes de MsDos, permitiendo la gestión remota. Se describará en profundidad en el capítulo 9.

obtener_hora: Con el objetivo de mantener a los clientes sincronizados y con una hora correcta (ahorra muchos problemas en la gestión del correo electrónico), se ha creado está función cuya misión es conectarse al puerto de *time* (puerto 37) del ordenador local, que estará redirigida por el cliente SSH al puerto de *time* (puerto 37) del servidor Linux.

El servidor devuelve un entero que representa el número de segundos transcurridos desde 1900. Se transforma este valor a la representación habitual de día, mes, año, hora, minuto, etc y se establece como hora actual del cliente. Para ello necesitaremos el número de segundos transcurridos desde 1970 y no desde 1900. Por ello habrá que restar al valor que nos da el servidor, el número de segundos transcurridos entre estos dos años: 2208988800.

La función descrita tiene el siguiente pseudocódigo:

```
creamos socket TIEMPO;
mientras canal SSH activo
        intento de conexión en el puerto 37 de localhost (redirigido por SSH);
si conexión lograda
        leemos doble palabra (4 octetos) del socket TIEMPO;
        convertimos doble palabra del orden de red a orden hardware;
        restamos 2208988800 para normalizar;
        creamos estructura de fecha y hora;
        ponemos nueva fecha y hora;
cerramos socket TIEMPO;
```

Para aumentar el rendimiento del sistema, hay que tener presente como funciona el enlace radio AX.25. Lo ideal es que los paquetes que se transmitan sean de tamaño máximo posible, de tal forma que se necesitan menos confirmaciones por bytes enviados (esto sólo es válido en enlaces radio con tasas de error bajas).

En pruebas preliminares se comprobó que el socket SSH proporcionaba normalmente los datos en paquetes menores de 256 bytes, que es el tamaño máximo de la parte de dados permitidos por el protocolo AX.25. Por ello, se implementó un

tiempo de espera en el socket SSH durante el cual se esperaba hasta llenar un paquete de 256 bytes. Obviamente este tiempo tiene que ser suficientemente pequeño para que no afecte en los paquetes que necesariamente son menores de 256 bytes. Así, definimos un plazo de 100mseg:

```
#define SOCKETTIMEOUT 100000;
```

En las fases en las cuales el cliente y servidor están intercambiando continuamente información (fase de autentificación SSH, autentificación POP3, listado y borrado de mensajes), el tamaño de los paquetes es normalmente menor que 256 bytes, por lo que no es conveniente esperar el tiempo establecido anteriormente. Para conseguir esto, sólo entrará en funcionamiento la espera si se han transmitido un mínimo de 5 paquetes sin haber recibido ninguno (durante la transmisión del mensaje sólo hay datos en el socket en una dirrección).

La función que se encarga de este almacenamiento está incluida en la línea de pseudocódigo leer socket SSH y tiene el siguiente esquema:

BytesLeidosTotales = 0;
esperar datos en Socket SSH con TIMEOUT;
si datos en socket SSH
    leer $(256 - BytesLeidosTotales)$ bytes;
    almacenar en $(buffer + BytesLeidosTotales)$;
si TIMEOUT o BytesLeidosTotales=256
    escribir buffer en canal AX.25;

6.4. Funciones auxiliares

ax25recv. Se encarga de recibir paquetes del socket de comunicación con AGW. Por este socket llegan paquetes en el siguiente formato (ver también Anexo F):

Offset	Variable
+0	Puerto
+1	Reservado
+4	Datakind
+5	Reservado
+8	Indicativo origen
+18	Indicativo destino
+28	Longitud de los Datos
+32	Reservado
+36	Datos

Figura 13: Estructura de los *frames* AGW

ax25send. Esta función manda paquetes en formato AGW para enviar datos por el canal AX.25 Usa el mismo tipo de *frames* que la función anterior.

log. Para llevar un seguimiento de las conexiones y cantidad de datos enviado por el cliente, el *daemon* crea un fichero de *logs* donde se almacena esta información. Se hace mediante el sistema de *logs* rotatorios que evitan un crecimiento progresivo.

fatal. Cuando se produce un error fatal, que detiene el funcionamiento del *daemon*, ésta es la función ejecutada. Muestra por pantalla el error fatal que ha llevado a parar el *daemon*. Las razones por las que puede finalizar su ejecución son las siguientes:

- No se encuentra el programa AGW.
- Error en la inicialización de la biblioteca de sockets de Windows (WSA).
- No se encuentra el programa SSH32.
- Errores en las funciones de sockets: *socket*, *bind*, *listen* y *accept*.
- Error al crear las hebras.
- No se encuentra el fichero de configuración. `daemon.cfg`, donde se almacena la información de configuración.
- El cliente SSH no está correctamente configurado.

6.5. Configuración

6.5.1. Configuración de AGW

Cuando el programa AGW es activado aparece un icono en el *system tray* (parte inferior-derecha). El menú de *Properties* tiene una primera sección donde se configura el módem (figura 14).

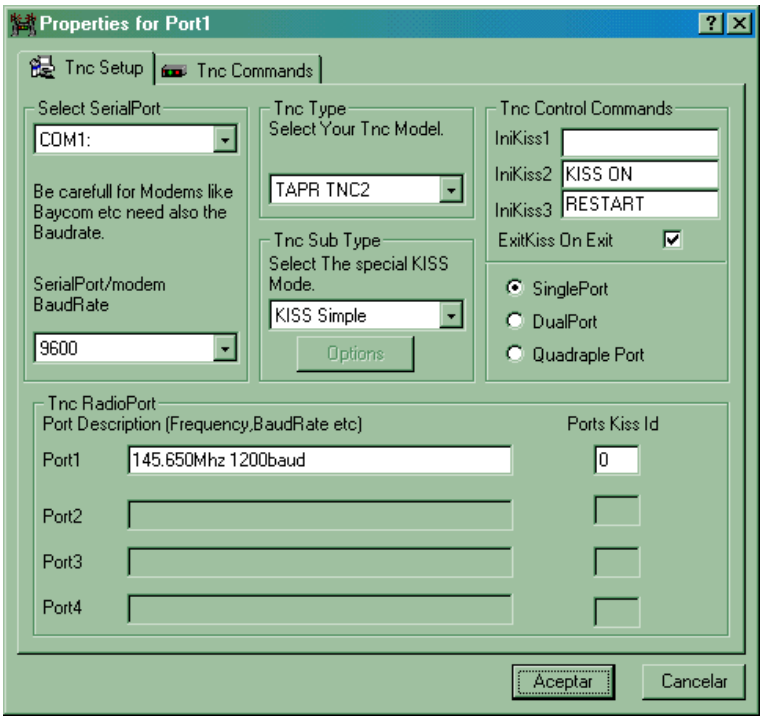


Figura 14: Opciones de AGW. Configuración del módem

Select Serial Port: `COM1`. Si se usan módems conectados al puerto serie, aquí se pone en que puerto se encuentra. En el caso de la tarjeta de sonido aquí ponemos el puerto que controla el PTT de la radio. En módems conectados al puerto paralelo este parámetro carece de importancia.

TNC Type: Selección del módem. Los cuatro analizados este proyecto han sido:

- `TAPR TNC2`
- `PIC-PAR`
- `YAM`
- `Sound Card`

Serial Port Baud Rate: Con este parámetro se especifica la velocidad de comunicación con el módem. Se corresponde con la velocidad de transmisión en el aire excepto en el caso de TNCs.

Si se trata de una TNC esta velocidad se refiere a la existente entre la TNC y el ordenador, debiendo ser como mínimo el doble que la del aire. Lo habitual es poner 9600bps cuando se transmite a 1200bps, y configurar 19200bps cuando se transmite a 9600bps. La TNC tiene que estar configurada para poder trabajar a la velocidad seleccionada (esta selección se hace manualmente con *jumpers* en la parte de abajo de la TNC)

El segundo menú de configuración de AGW hace referencia a parámetros de acceso al medio y *timeouts* (plazos) del nivel AX.25 (figura 15).

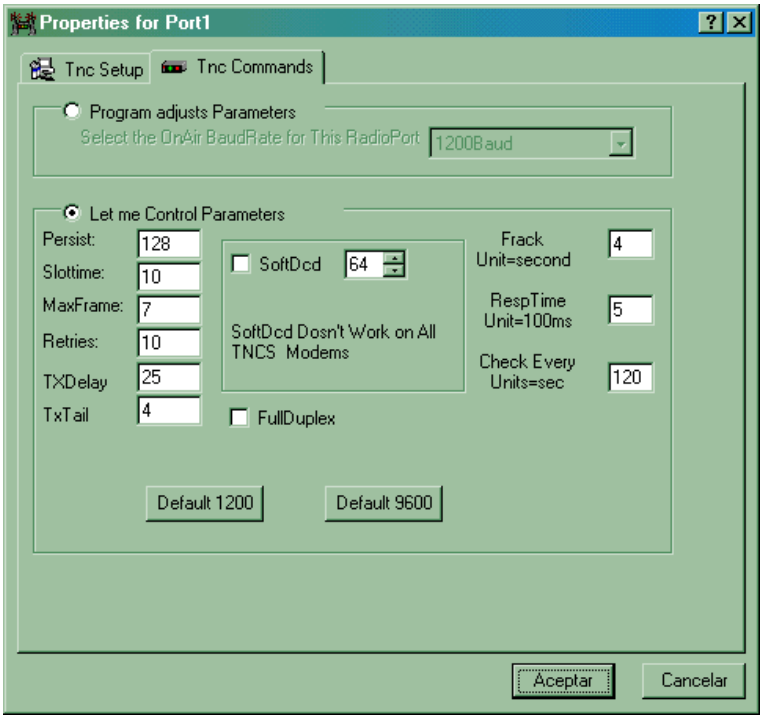


Figura 15: Opciones del cliente SSH

Persist: 128. El acceso al medio en el enlace radio es CSMA p-psersistente. En el momento en que el módem tiene un paquete a transmitir, en vez de enviarlo siempre, sólo lo envía con una probabilidad=$Persist/256$. En el caso de encontrar el canal ocupado se espera un tiempo (*Slot Time*) y vuelve a intentar enviar con la probabilidad anterior. Y continúa así hasta

que es capaz de enviarlo. Si el protocolo DAMA no está activo hay que poner un valor bajo (p.e. 128) para evitar excesivas colisiones, más bajo cuanto más estaciones formen la subred. Si en caso contrario disponemos del protocolo DAMA, se puede poner un valor mucho más alto (p.e. 250), pues la posibilidad de colisión de paquetes es mucho menor.

Slottime: Es el tiempo descrito anteriormente. Se suele poner un valor de 100 mseg (se pone 10 por expresarse en decenas de *mseg*).

MaxFrame: El protocolo AX.25 establece una ventana de 7 paquetes como máximo para transmitir sin recibir confirmación. Este conjunto de paquetes que se transmiten antes de esperar confirmación recibe el nombre de *frame*. Al no disponer de retransmisiones selectiva de paquetes perdidos, las partes retransmiten todos los paquetes empezando por el que se ha producido el error, aunque los posteriores hayan llegado correctamente. Por ello es recomendable disminuir el valor de este parámetro para enlaces radio con tasas de error elevadas.

Retries: Máximo número de retransmisiones de un paquete de demanda de estado antes de cerrar el canal por pérdida del enlace.

Txdelay: Este parámetro depende por completo de la radio que estemos usando. Es el tiempo que hay que tener el PTT activo (nivel bajo) antes de empezar a transmitir un paquete. El PTT (*Push To Talk*) es la entrada de la radio que selecciona si recibimos (nivel alto) o transmitimos (nivel bajo). Este tiempo es muy relevante en radios *half-duplex* pues el tiempo de commutación entre recepción y transmisión es grande (alrededor de los 250 mseg).

TxTail: Es el tiempo que hay que mantener el PTT a nivel bajo después de haber enviado a la radio el último de los datos.

Frack: *Frame Acknowledge*. Este es el tiempo de reconocimiento de *frame*. Si se supera este tiempo desde la transmisión de un paquete sin confirmación, se envía una petición de estado al interlocutor. Este tiempo se suele denominar también *Timeout 1*.

RespTime: Tiempo de Respuesta. Marca el máximo periodo de espera antes de enviar una confirmación a la otra parte (sin que este la haya pedido). Este tiempo se suele denominar *Timeout 2*.

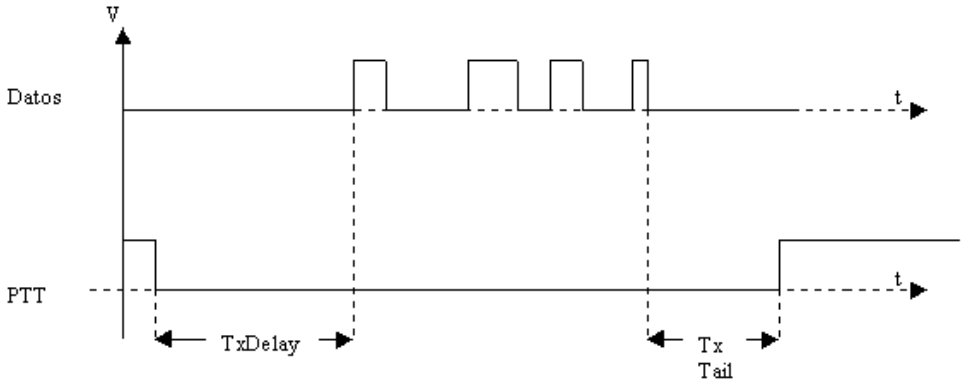


Figura 16: Cronograma de la transmisión de un paquete de datos

Check: Tiempo de chequeo. Superado este tiempo sin que se haya enviado o recibido ningún paquete, se pide una confirmación a la otra parte de que el enlace aún está activo. Se evita, pues, mantener recursos para enlaces en los que alguna de las partes ha caído o ha desconectado. Este tiempo se suele denominar también *Timeout 3*.

En el Anexo D se puede encontrar la descripción sobre la importancia y modo de ajuste correcto de estos parámetros.

6.5.2. Configuración de SSH32

La pantalla de configuración de SSH es la que vemos en la figura 17, en ella tenemos que configurar lo siguiente:

- **Profile Name**: `holden`. Este es el *Profile Name* (nombre de perfil) que el *daemon* usa por defecto. Este nombre es totalmente arbitrario, puede ponerse el que se quiera mientras el *daemon* cliente invoque al cliente SSH con este nombre como parámetro.
- **Host Name**: `localhost`. El cliente debe conectarse a nuestro *daemon*, activo en la máquina local.
- **User ID**: `ssh_user`. Hay que sustituir esta entrada por el nombre del usuario en el servidor. Esta cuenta será la que se usará para la autentificación, después se podrá leer el correo del usuario que se desee, según la configuración del cliente de correo.

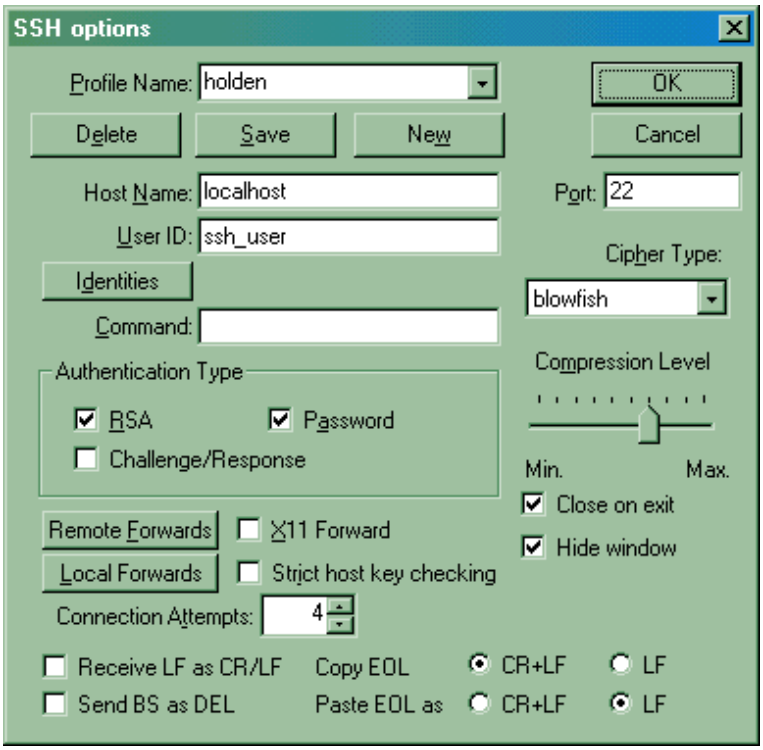


Figura 17: Opciones del cliente SSH

- **Port**: 22. Este es el puerto habitual de los servidores SSH. Nuestro *daemon* por defecto escucha en ese puerto, así que no hay que modificarlo.
- **Cypher Type**: `Blowfish`. Aquí hay que poner el tipo de cifrado deseado. Las opciones son: IDEA, Blowfish, DES, 3DES, Arcfour o ninguno. Se ha escogido Blowfish por tener una seguridad elevada y la velocidad de codificación más rápida de todas las opciones disponibles.
- **Authentication Type**: `RSA, Password`. El sistema escogido de autenticación es RSA, basado en un sistema de claves públicas y privadas. La pública (extensión `.PUB`) se almacena en el servidor y la privada la guarda confidencialmente cada uno de los clientes. La opción `Password` hay que mantenerla activa porque en un primer momento no estarán generadas las claves públicas y privadas, así que la autentificación se hará mediante la introducción de una contraseña, que será la que tenga el usuario en la máquina servidor.
- **Compression Level**: 6. El nivel de compresión tiene que guardar un compromiso entre tasa de compresión y retardo de envío. Dicho retardo aumenta con tasas de compresión mayores. Un valor de 6 ha mostrado un comportamiento adecuado.

- **Close on exit**: `Activado`. Cuando el *daemon* cierre el socket, con esta opción automáticamente también se cerrará el SSH32.
- **Hide Window**: `Activado`. Cuando se ha acabado el proceso de autentificación el SSH32 desaparece de la barra de tareas y pasa al *system tray*.
- **Local Forwards**: En esta ventana hay que definir los puertos locales que van a ser redireccionados. En nuestro caso serán los puertos 25000 (SMTP-bis), 11000 (POP3bis) y 37 (se usará para la sincronización de hora de los clientes).

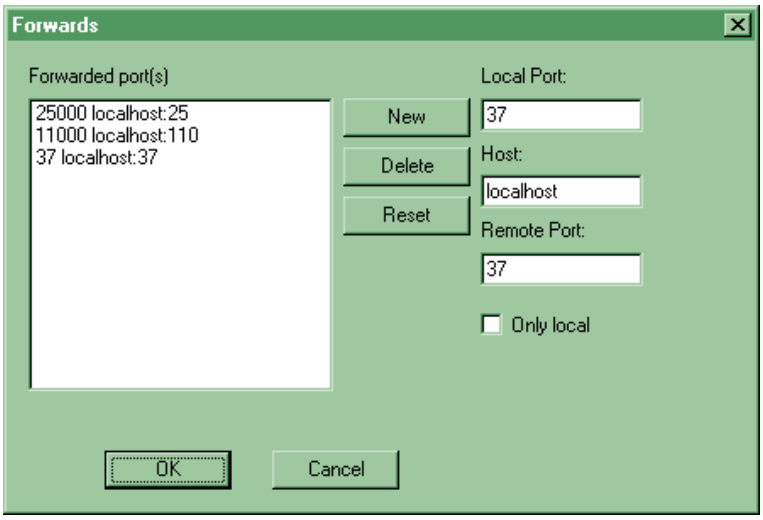


Figura 18: Opciones del cliente SSH: Redirección de puertos

La sintaxis de un puerto redireccionado (*forwarded port*) es la siguiente:

- **Forwarded port**: `Puerto_Local Host:Puerto_Remoto`.

Los puertos que han sido redireccionados son los siguientes:

- **Forwarded port**: `25000 localhost:25`.
- **Forwarded port**: `11000 localhost:110`.
- **Forwarded port**: `37 localhost:37`.

Donde 25 y 110 son los puertos de SMTP y POP3, 25000 y 11000 los puertos denominados SMTPbis y POP3bis. El puerto 37 se corresponde con el puerto de TIME de un ordenador Linux.

El propio cliente SSH permite la generación del par de claves pública/privada. En la instalación final, esto se hará de forma automática. Manualmente se haría `Keys` →`Generate Key`.

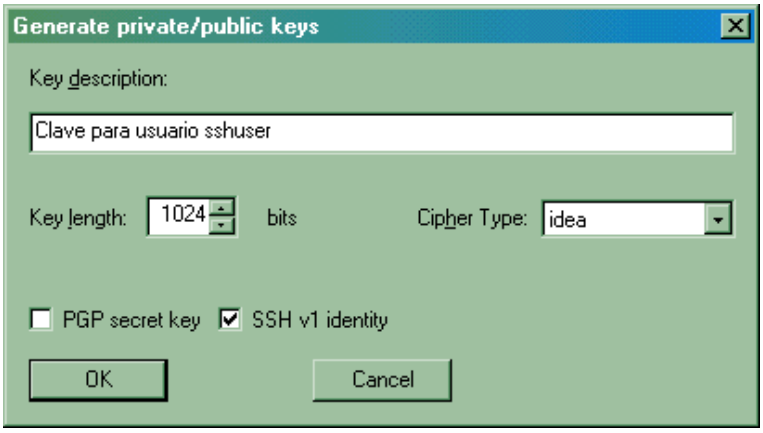


Figura 19: Opciones del cliente SSH: Generación de claves

- **Key description**: `Clave RSA`. Puramente informativo.
- **Key length**: `1024`. Longitud de la clave RSA. 1024 es un tamaño adecuado.
- **Cipher type**: `Idea`. Este parámetro define el tipo de cifrado en la autentificación. IDEA es el más usado por la mayoría de servidores SSH.
- **PGP secret key**: `Desactivado`.
- **SSH v1 identity**: `Activado`. Las claves generadas son válidas para servidor SSH con versión 1.x.

Ya podemos presionar `OK` y nos pedirá el nombre del fichero que contendrán las claves, por ejemplo `sshuser`. Posteriormente, nos pedirá la introducción de una *passphrase* (similar a una contraseña). Lo dejamos vacío para que la autentificación se haga de forma automática sin intervención del usuario.

Ahora hay que añadir la clave privada que acabamos de generar (`sshuser`) en la lista de identidades del usuario. Para ello volvemos a la pantalla de opciones y en menú `Identities` hacemos lo siguiente:

Identities →New (seleccionar `sshuser`) →OK →Save

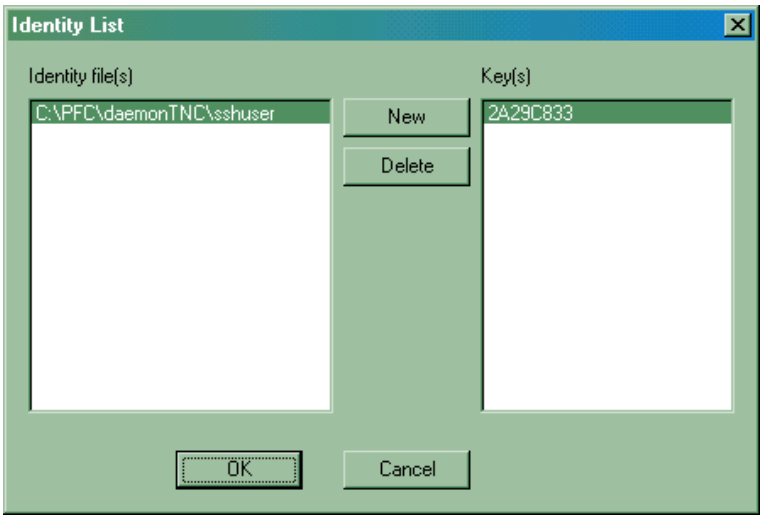


Figura 20: Opciones del cliente SSH: Selección de claves

6.5.3. Configuración del *daemon*

El *daemon* mantiene la configuración en el fichero `daemon.cfg` del directorio donde se haya instalado el paquete con el *daemon*. Los campos que forman el fichero de configuración es la siguiente:

`ID4CLI` ←*Indicativo del cliente*
`ID4ABC` ←*Indicativo del servidor*
`ehaspass` ←*Contraseña para gestión remota*

6.6. Instalación y configuración

En primer lugar, lo primero que hay que destacar es que el protocolo TCP/IP debe estar instalado en el ordenador. Como lo normal es que éste no disponga de ningún dispositivo de red sobre el que instalar TCP/IP, lo más sencillo es instalar el Acceso telefónico a redes como Adaptador de red y posteriormente TCP/IP ligado a él.

Respecto al proceso de instalación del *daemon*, éste ha sido automatizado mediante el uso de un instalador *freeware* de paquetes Windows (Setup Generator) y un interfaz gráfico de configuración (figura 23), que evita la configuración manual del SSH y del fichero de configuración (`daemon.cfg`).

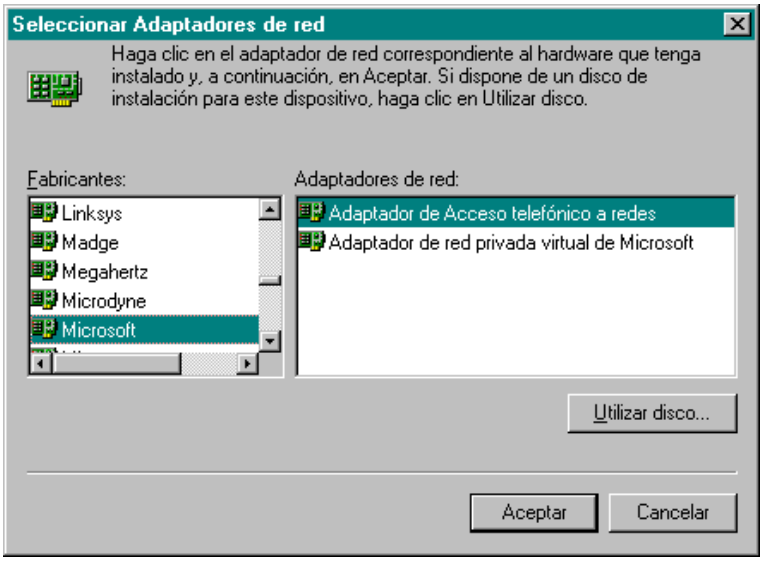


Figura 21: Instalación del acceso telefónico a redes

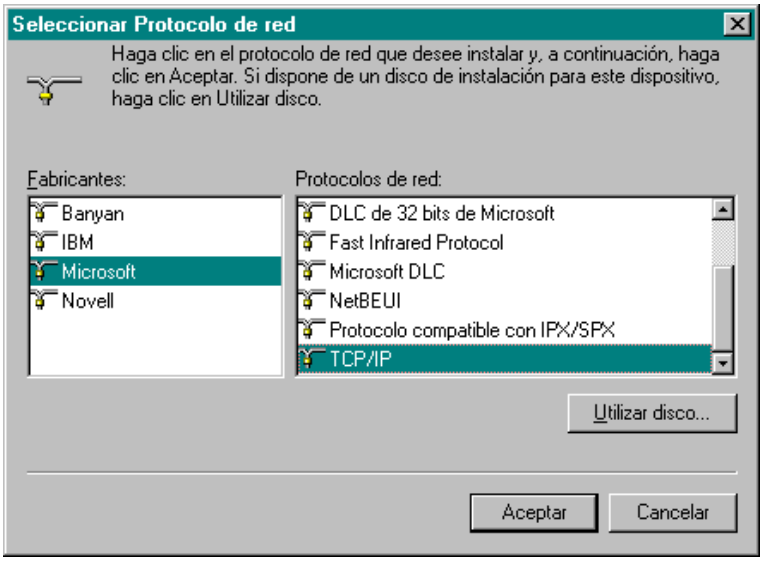


Figura 22: Instalación del protocolo TCP/IP

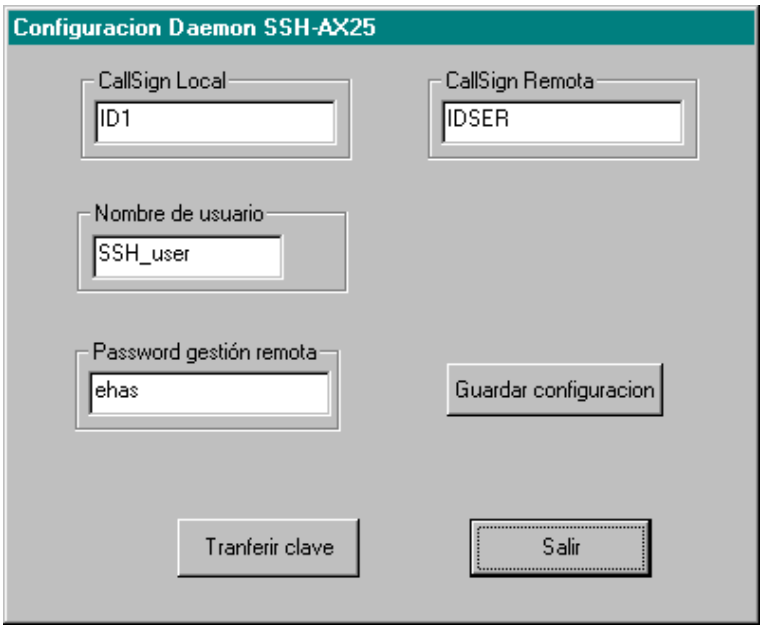


Figura 23: Opciones del *daemon* de Windows

Una vez ejectuado y finalizado el programa de instalación generado (*WinDaemon.exe*) se crea la carpeta con los accesos directos al *daemon* (figura 24) y se abre una ventana de configuración con los siguentes parámetros:

- **CallSign Local**: Indicativo del cliente
- **CallSign Remota**: Indicativo del servidor
- **Password de gestión remota**: Contraseña para entrar en el cliente Windows desde un servidor Linux.
- **Nombre usuario SSH**: Nombre del usuario en cuya cuenta se autentificará el cliente SSH32.
- **Guardar configuración**: Se guarda la configuración modificando en el fichero `daemon.cfg` los indicativos y la contraseña, y modificando el registro donde se almacena la configuración del SSH32. Si la clave para ese usuario SSH no existía se crea. La creación de las claves se hace con el programa `SSH-KEYGEN.EXE`, de distribución gratuita.
- **Transmitir clave**: Se transfiere la clave pública (*[nombre_de_usuario.PUB]* al servidor. La transferencia se hace gracias al uso del programa SCP (Secure Copy, ver sección A.5). Para que la conexión sea posible se activa el *daemon* mientras dure la transferencia.

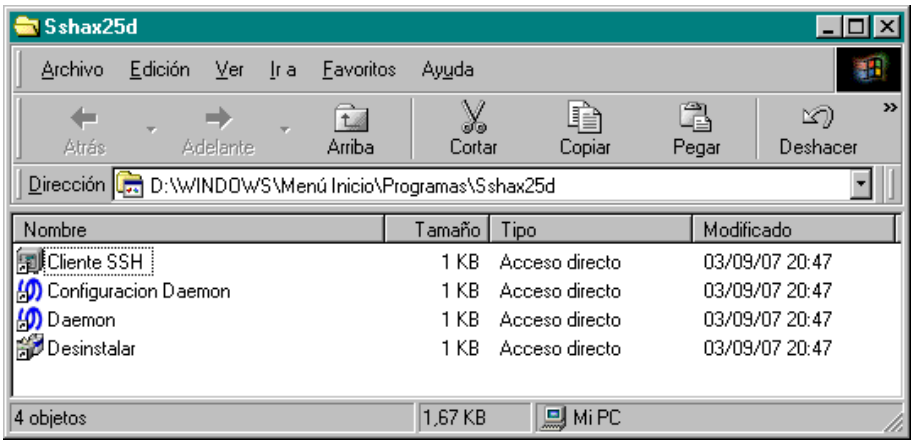


Figura 24: Carpeta del *daemon* de Windows

Una vez finalizado este proceso de instalación y configuración el *daemon* debería quedar instalado en la clave registro de Windows que arranca programas al inicio de sesión. Para hacerlo hay que añadir el path del *daemon* en la clave de registro:

`HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run`

También se incluye un icono para desinstalar la aplicación completa, excepto los ficheros que se hayan añadido posteriormente (las claves públicas y privadas), cuyo borrado es opcional.

6.6.1. Configuración de Netscape

En el Messenger hay que poner la siguiente configuración:

Servidor POP3 entrada. localhost.

Usuario POP3. El usuario de correo en el servidor.

Servidor SMTP salida. localhost.

Es conveniente deshabilitar la opción *Comprobar correo cada X minutos* para evitar accesos a radio no pedidos.

Si se ha realizado todo el proceso con éxito, una vez rearrancado el ordenador, habrá aparecido un nuevo icono en el *system tray* (se trata del AGW) y habrá un proceso residente (el *daemon*) que se puede ver si está activo haciendo Ctrl-Alt-Del, debiendo aparecer en la lista de procesos.

En el momento de dar a **Obtener Mensaje** o **Enviar Mensaje** aparecerá en la barra de tareas el SSH durante la fase de autentificación. En el momento en que desaparezca de la barra de tareas y aparezca en el *system tray* querrá decir que el proceso de autenticación ha finalizado y ya se ha establecido la comunicación con los puertos SMTP ó POP3 del servidor. Una vez la la comunicación finalice, el icono de SSH también desaparecerá.

7. Implementación en el servidor

7.1. Introduccion

El servidor tendrá el sistema operativo Linux/Debian, que ya dispone en su distribución de todas las aplicaciones necesarias para funcionar como servidor de correo. De igual forma que en la parte Windows, habrá que programar un *daemon* que haga la función de pasarela entre el enlace radio y el servidor SSH.

7.2. Funcionamiento del *daemon* servidor (sshax25d)

El *daemon* para el servidor Linux se ha programado con el compilador `gcc` estándar. El *daemon* está escuchando en los puertos AX.25 por si se produce alguna petición de conexión, momento en el cual sirve de pasarela entre los datos que llegan del enlace radio (socket AX.25) y el socket con el servidor SSH, socket que habrá abierto al recibir la petición. A diferencia del *daemon* cliente, el de la parte del servidor no tiene que preocuparse de las conexiones que se produzcan en los puertos SMTP ó POP3 pues la información referente a estas conexiones va en el túnel SSH y debe ser el servidor SSH el encargado de abrir y gestionar las conexiones con los puertos de correo. Así pues, el protocolo de actuación del *daemon* servidor es el siguiente:

1. El *daemon* está a la espera de conexiones del socket AX.25 ligado (función *bind*) a su indicativo.

2. Cuando se produce una conexión por parte de algún cliente, se acepta la petición y se hace una conexión con el puerto del servidor SSH (puerto 22). En este punto el programa se bifurca en dos procesos, mediante la orden `fork`, uno de ellos procesa la petición y el otro se pone en espera de una nueva petición de otro cliente.

3. Una vez el servidor SSH contesta a nuestra petición de conexión, ya puede empezar el intercambio de información entre el socket AX.25 y el socket SSH, mediante la función `select` de la biblioteca estándar de *sockets*.

4. Cuando el cliente ha terminado la lectura o escritura en los puertos de correo, manda una petición de desconexión. Nuestro *daemon* ya puede cerrar la conexión con el servidor SSH y se finaliza el proceso.

El código completo del *daemon* se puede encontrar en el disco que acompaña el proyecto (*sshax25d.c*). Las operaciones que realiza el *daemon* son las siguientes:

```
crea socket AX.25;
bind socket AX.25;
listen socket AX.25;
bucle infinito
        si recibida llamada AX.25
                bifurcación (fork);
                si es el proceso padre vuelve inicio bucle infinito;
                crea socket SSH;
                connect socket SSH
                mientras Socket SSH y Socket AX.25 abiertos
                        esperar hasta que haya datos para leer;
                        si datos en socket SSH
                                leer socket SSH;
                                escribir socket AX25;
                        si datos en socket AX25
                                leer socket AX25;
                                escribir socket SSH;
                cerrar Socket SSH;
                cerrar Socket AX.25;
                finaliza proceso;
fin bucle infinito
```

Al igual que en el *daemon* para Windows, se ha hecho necesario un sistema de *buffering* intermedio que asegure el tamaño máximo de paquetes (si es posible) con el fin de maximizar la eficiencia del canal.

```
BytesLeidosTotales = 0;
esperar datos del Socket SSH Server con TIMEOUT;
si datos en socket SSH Server
        leer (256 − BytesLeidosTotales) bytes;
        almacenar datos leidos en (buffer + BytesLeidosTotales);
si TIMEOUT o BytesLeidosTotales=256
        escribir buffer en canal AX.25
```

7.3. Funciones auxiliares

void catch_child(int sig_sum). Se encarga de recibir el código de salida de los hijos que han aceptado una conexión y han finalizado. De esta forma se evita que queden procesos *zombies*.

7.4. Configuración

7.4.1. Configuración de AX.25

Configuración del kernel La configuración de AX.25 en Linux requiere la recompilación del kernel. Es adecuado usar un kernel con versión mayor o igual al 2.2.18 pues en esta versión se solucionaron algunos problemas de pérdida de memoria (*memory leakage*) de versiones anteriores. Una vez descomprimido el código fuente del kernel (por ejemplo `/usr/src/local/linux`) se debe configurar para poder usar AX.25 y el módem a usar (KISS, PicPar, YAM, etc).

```
make menuconfig
```

Si informa de algún error respecto a la falta de biblioteca, seguramente habrá que instalar la biblioteca *ncurses*. Ahora hay que configurar:

```
Code maturity level options
    [*] Prompt for development and/or incomplete code/drivers
Loadable module support
    [*] Kernel module loader
General Setup
    [*] Networking support
    [M] Parallel port support
        [M] PC-Style hardware
Amateur Radio support
    [*] Amateur Radio Support
    [M] Amateur Radio AX.25 Level 2 Protection
    [M] PicPar
```

Donde:
[*] Forma parte del kernel
[M] Instalado como módulo

Y ya se puede compilar el kernel como de costumbre:

Debian

```
make-kpkg clean
make-kpkg --revision ax25_1.0 kernel_image
dpkg -i kernel-image-2.2.18_ax25_1.0_i386.deb
```

Y rearrancar.

Instalación de paquetes También hay que instalar las aplicaciones y bibliotecas que se usan en Amateur Radio:

Debian

```
apt-get install ax25-tools ax25-apps libax25-dev
```

(automáticamente se instalarán también las libax25)

Ficheros de configuración Para poder usar un dispositivo como módem en AX.25 necesitamos configurarlo en el fichero `/etc/ax25/axports` [3]. Deberemos incluir las líneas necesarias que se vayan a usar, la TNC para la redes antiguas a 1200bps y Picpar para las nuevas redes a 9600bps:

/etc/ax25/axports

#puerto	#Indicativo	#Vel.	#Tamaño	#Ventana	#Descrip.
tnc	IDTSER	9600	255	7	TNC a 1200bps
picpar	IDPSER	9600	255	7	Picpar a 9600bps

Donde:

Puerto. Nombre del puerto que después se usará para establecer conexiones por el enlace radio

Indicativo. Indicativo del servidor.

Vel. Velocidad de comunicación con el módem. En todos los dispositivos *hardware* hay que poner la velocidad de transmisión en el aire, excepto en la TNC, donde hay que poner una velocidad al menos dos veces superior.

Tamaño. En este parámetro especificamos el número máximo de octetos de usuario que contiene los paquetes AX.25 a enviar.

Ventana. Tamaño de la ventana de transmisión, es decir, el máximo número de paquetes que se pueden transmitir sin recibir confirmación. El valor máximo es 7.

Descrip Descripción del puerto a título informativo.

Ahora creamos unos *scripts* de arranque para que se ejecuten de forma automática al arrancar el servidor. Para ello Linux dispone del directorio `/etc/init.d`. Los scripts que se encuentren en este directorio son llamados por unos enlaces simbólicos que se encuentran en los directorios `/etc/rcX.d` (Donde **X** son los niveles de usuario del 0 al 6). Para la creación automática de los enlaces existe la utilidad `update-rc.d`:

```
update-rc.d nombre_de_script defaults
```

El *script* carga los módulos del protocolo AX.25, los módems y el *daemon* **sshax25d**. Además asigna valores a las constantes que rigen el protocolo AX.25 que se encuentran en `/proc/sys/net/ax25/[disp]` [3]:

- Red antigua con TNCs a 1200bps:
  - Mandamos dos secuencias para asegurar que la TNC esté en modo KISS: `'kiss on'` y `'restart'`.
  - TNC conectada en puerto serie **COM1**.
  - *txdelay* = 350 mseg, *txtail* = 150 mseg. (radios Motorola PRO3100)
  - *slottime* = 10 mseg, *ppersist* = 128.
  - Tamaño de paquete = 256, Tamaño de ventana = 7.
  - *Timeout1* = 15 seg. Este tiempo debe ser mayor al tiempo mínimo de envío de una ventana de 7 paquetes. En caso contrario el PC pediría una retransmisión antes de que la TNC hubiera sido capaz de enviar toda la ventana. El tiempo mínimo es 7x256x8/1200 = 12 segundos.

- *Timeout2* = 5 segundos.
- *Timeout3* = 3 segundos.
- *Idletimeout* = 10 minutos.

ax25tnc

```
# /bin/sh
DAEMON=/usr/local/sbin/sshax25d
NAME=sshax25d
DESC="SSH-AX.25 Daemon"
case "$1"in
  start)
    echo -e -n "\rkiss on\r"> /dev/ttyS0
    echo -e -n "restart\r" > /dev/ttyS0
    insmod ax25
    insmod mkiss
    /usr/sbin/kissattach /dev/ttyS0 tnc 44.0.0.100
    /usr/sbin/kissparms -p tnc -t 350 -s 10 -r 128 -l 150
    echo 256 > /proc/sys/net/ax25/ax0/maximum_packet_length
    echo 7 > /proc/sys/net/ax25/ax0/standard_window_size
    echo 2000 > /proc/sys/net/ax25/ax0/t1_timeout
    echo 500 > /proc/sys/net/ax25/ax0/t2_timeout
    echo 3000 > /proc/sys/net/ax25/ax0/t3_timeout
    echo 60000 > /proc/sys/net/ax25/ax0/idle_timeout
    set -e
    echo -n "Starting $DESC: "
    start-stop-daemon --start --pidfile /var/run/$NAME.pid -b
      --make-pidfile --exec $DAEMON
    echo "$NAME."

  stop)
    echo -n "Stopping $DESC: "
    start-stop-daemon --stop --quiet --pidfile /var/run/$NAME.pid
      --exec $DAEMON
    echo "$NAME."
    ;;
esac
exit 0
```

- Nueva red a 9600bps con módems PicPar y DAMA (ver próximo capítulo):
  - Picpar conectado en puerto paralelo LPT1 (puerto 0x378, irq 7).
  - *txdelay* = 150 mseg, *txtail* = 20 mseg (radios Yaesu VX-2000).
  - *slottime* = 10 mseg, *ppersist* = 230.
  - Tamaño de paquete = 256, Tamaño de ventana = 7
  - *Timeout1* = 4 seg. Este tiempo debe ser mayor al tiempo mínimo de envío de una ventana de 7 paquetes. En caso contrario el PC pediría una retransmisión antes de que el PicPar hubiera sido capaz de enviar toda la ventana. El tiempo mínimo es 7x256x8/9600 = 1.5 segundos.
  - *Timeout2* = 500 mseg.
  - *Timeout3* = 6 segundos.
  - *Idletimeout* = 10 minutos.
  - *protocol* = 3. Se activa DAMA (ver próximo capítulo)
  - *dama_master_timeout* = 10 seg. Tiempo de turno de DAMA.

ax25picpar

```
DAEMON=/usr/local/sbin/sshax25d
NAME=sshax25d
DESC="SSH-AX.25 Daemon"
case "$1"in
  start)
    insmod ax25
    insmod parport
    insmod parport_pc io=0x378 irq=7
    modprobe baycom_par mode="picpar"iobase=0x378
    /usr/sbin/sethdlc -p -i bcp0 mode "picpar"io 0x378
    /usr/sbin/sethdlc -a -i bcp0 txd 150 txtail 20 slot 100
      ppersist 230
    ifconfig bcp0 44.0.0.101 up
    ifconfig bcp0 hw ax25 ID2PIC up
    echo 7 > /proc/sys/net/ax25/bcp0/standard_window_size
    echo 400 > /proc/sys/net/ax25/bcp0/t1_timeout
```

```
      echo 500 > /proc/sys/net/ax25/ax0/t2_timeout
      echo 600 > /proc/sys/net/ax25/bcp0/t3_timeout
      echo 60000 > /proc/sys/net/ax25/bcp0/idle_timeout
      echo 3 > /proc/sys/net/ax25/bcp0/protocol
      echo 10 > /proc/sys/net/ax25/bcp0/dama_master_timeout
      set -e
      echo -n "Starting $DESC: "
      start-stop-daemon --start --pidfile /var/run/$NAME.pid -b
        --make-pidfile --exec $DAEMON echo "$NAME."
      echo "$NAME."

    stop)
      echo -n "Stopping $DESC: "
      start-stop-daemon --stop --quiet --pidfile /var/run/$NAME.pid
        --exec $DAEMON
      echo "$NAME."
      ;;
  esac
  exit 0
```

En este punto hay que volver a destacar que la IP que ponemos no se usa en ningún momento en nuestro sistema. Ya no estamos encapsulando TCP/IP en AX.25, solamente nos estamos comunicando con puertos TCP de la propia máquina (localhost). La IP hay que ponerla porque la implementación que han hecho así lo obliga pero no la usaremos para nada. Podríamos poner la misma en todas las estaciones sin ningún problema.

7.4.2. Configuración de SSH

En el servidor Linux/Debian instalamos el paquete `ssh` que forma parte de la distribución Debian:

```
apt-get install ssh
```

La instalación del paquete incluye un cliente y un servidor. El servidor queda instalado en `/usr/sbin/sshd` y el cliente en `/usr/sbin/ssh`. El *daemon* `sshd` se inicia cada vez que arrancamos el ordenador.

7.5. Administración de cuentas

Las cuentas de usario se crean de la forma habitual (*adduser)*. Para que la autentificación mediante el sistema de claves públicas sea posible debe existir un directorio `.ssh` en el directorio raíz de cada uno de los usuarios. Dentro de este directorio el servidor SSH mirará las claves públicas almacenadas en el fichero `authorized_keys`.

Con el sistema automático de transferencia de claves visto en el capítulo anterior, la clave pública es copiada automáticamente en el fichero **authorized_keys**.

8. DAMA

Con la implementación de DAMA realizada en este proyecto se ha alcanzado la consecución de los siguientes objetivos:

- Posibilidad de instalación de redes en emplazamientos de orografía complicada, necesitando únicamente visibilidad directa entre cliente y servidor.
- Instalación de antenas directivas en los clientes, con el consiguiente ahorro de energía y posibilidad de instalación de antenas de menor altura.
- Aumento de la eficiencia del canal radio, debido a la eliminación de colisiones de paquetes, que ya sólo serán posibles en la fase de petición de acceso al servidor.

La implementación del protocolo DAMA [8] que se ha realizado hasta ahora por grupos de radioaficionados requiere la modificación tanto del servidor como del cliente. Esto nos planteaba dos alternativas a la ahora de encarar nuestra propia implementación:

Aplicación AX.25-DAMA para Windows. Como no existe ningún programa para Windows que incorpore DAMA al protocolo AX.25 (ni AGW ni Flexnet lo hacen), deberíamos programar un sistema completo que realizara las funciones de AX.25-DAMA. Tal programa es de una complejidad considerable, fuera del alcance del presente proyecto.

Uso de AGW en Windows. Si bien el protocolo DAMA tal como está especificado requiere cambios en el cliente, se podría realizar de una forma alternativa de tal forma que un cliente normal que sólo soportara el protocolo AX.25 pudiera ser usado dentro de una red controlada por un servidor DAMA. De esta forma sólo se requerirían modificaciones en el servidor, al que tenemos acceso completo por tratarse de un sistema Linux y disponer del código fuente.

La opción escogida fue la segunda, con tal de poder seguir usando un programa como el AGW para los clientes, conservando las ventajas que tiene su uso y que ya fueron descritas.

Así pues, sólo se han hecho modificaciones en el servidor Linux, lo que se ha traducido en una modificación del kernel.

8.1. Implementación en el servidor

8.1.1. Paquetes AX.25 usados

Para realizar la implementación sólo disponemos de los paquetes estándar del protocolo AX.25, pues éstos son los únicos que entiende AGW, el programa encargado del acceso al medio radio en Windows. Veamos cuales se han usado:

RNR. ***Receiver Not Ready.*** Este paquete se suele usar para indicar al interlocutor que nuestro buffer de entrada, normalmente de una TNC, está lleno y no podemos gestionar la llegada de nuevos paquetes. Independientemente de que el módem haga o no uso del paquete **RNR**, podemos usarlo para nuestro propósito. Cuando un cliente haya consumido el tiempo asignado a su turno, el servidor mandará este paquete y el cliente parará de transmitir.

RR. ***Receiver Ready.*** Este paquete tiene un uso más general dentro del protocolo AX.25. Sirve para confirmar la recepción de paquetes y para comprobar el estado de un enlace. Lo usaremos para indicar a un cliente que ya puede reanudar la transmisión y recepción de paquetes que había sido interrumpida por la recepción de un **RNR**.

8.2. Descripción del funcionamiento de DAMA

8.2.1. Establecimiento de conexión

Cuando un cliente intenta una conexión con el servidor, éste añade su indicativo en una lista. Si se trata de la única conexión activa en ese momento, el usuario dispone del canal de forma exclusiva y se continúa la comunicación de forma normal, respondiendo con una aceptación de conexión AX.25.

Si, en caso contrario, el usuario ha pedido conexión pero ya existían usuarios a los que el servidor estaba sirviendo, el nuevo cliente es añadido a la lista de conexiones activas, aceptada su petición, pero immediatamente también le envía un paquete RNR con lo que le niega la posibilidad de transmitir ni recibir ninguna información hasta que le llegue su turno.

En la fase de conexión es normal que se produzcan colisiones de paquetes, pues un cliente que intente una conexión no escucha los paquetes que en ese momento esté enviando el cliente activo al servidor.

8.2.2. Transferencia de datos cliente-servidor

Cuando un usuario agote el tiempo asignado por el servidor, éste le enviará un paquete RNR, con lo que el cliente no estará autorizado a enviar más datos. El siguiente cliente en la lista recibirá el paquete RR y se reanudá la comunicación en el punto que había sido interrumpida.

Hay que notar que aún hay posibilidades de que se produzcan colisiones de paquetes. Éstas se puede dar si el paquete de petición de cese de transmisión (RNR) se pierde, pues en ese momento hay dos clientes que piensan que tienen el turno asignado. Si esto se produce el servidor vuelve a enviar RNRs al cliente que ya había perdido el turno hasta que éste cesa su transmisión.

El tiempo de turno a cada cliente es una variable constante que se configura en el momento de cargar el *daemon*. Un valor adecuado en una red de 9600bps es de unos 10 segundos. No se ha diseñado un sistema de plazos en el caso de que un cliente no transmita porque nuestro sistema está preparado para la transmisión y recepción de correo electrónico, y en este tipo de comunicaciones ninguna de las partes detiene nunca la comunicación.

8.2.3. Desconexión

Una petición de desconexión (DISC), tanto por parte del cliente como del servidor, sólo estará permitida cuando el usuario correspondiente tenga el turno asignado, de forma que no se moleste a los otros usuarios. Cuando la petición de desconexión es aceptada por el servidor (UA), el usuario es eliminado de la lista de usuarios activos y se pasa a servir de forma inmediata al siguiente de la lista, mandándole el correspondiente paquete RR.

8.2.4. Control de inactividad

Si un cliente no contesta al servidor a la asignación de turno (RR) un determinado número de veces, hay que suponer que el enlace ha caído o el cliente se ha desconectado sin que el paquete correspondiente (DISC) haya llegado al servidor. En este caso el servidor lo desconecta y elimina de la lista, liberando todos los recursos que ocupaba.

8.2.5. Compatibilidad entre DAMA y CSMA

Con el sistema propuesto, la compatibilidad entre ambos sistemas es total, pues los clientes no requien ningún tipo de modificación: con este sistema una red en la cual el servidor central incorpore DAMA se convierte automáticamente en una red DAMA en su totalidad.

8.3. Implementación

Veamos unos diagramas de flujo donde se aprecia el protocolo de actuación del protocolo DAMA en el servidor. Tres son los casos a contemplar:

- Llegada de una petición de conexión (SABM). Figura 25
- Fin del temporizador DAMA (dama_timeout). Figura 26
- Llegada de una petición de desconexión (DISC). Figura 27

La implementación de AX.25 en Linux se encuentra en el propio kernel, así que hay que conseguir las fuentes de la versión que vayamos a usar y descomprimirlo, en nuestro caso usaremos la versión de kernel 2.2.18.

Los dispositivos que usan el protocolo AX.25 están integrados en Linux de forma que una vez configurados tienen su entrada en la lista de dispositivos de red y por tanto podemos realizar con ellos las operaciones típicas de *sockets* (*accept*, *connect*, *listen*, etc). Para la implementación de DAMA será necesario sólo modificar el nivel de protocolo AX.25. Los ficheros de interés son los siguientes:

include/net/ax25.h Fichero con constantes y defínción de estructuras.

include/linux/sysctl.h Mantiene el nombre de los ficheros de configuración que se instalan en `/proc/sys/net/` al activar un dispositivo de sistema. En nuestro caso nos interesan los que instalará en `/proc/sys/net/ax25` (ver Anexo C).

net/drivers Aquí están los ficheros que se encargan del control a bajo nivel de los módems. No será necesario hacer modificaciones.

net/ax25 En este directorio están los ficheros que implementa el protocolo AX.25. La mayor parte de modificaciones se realizarán aquí.

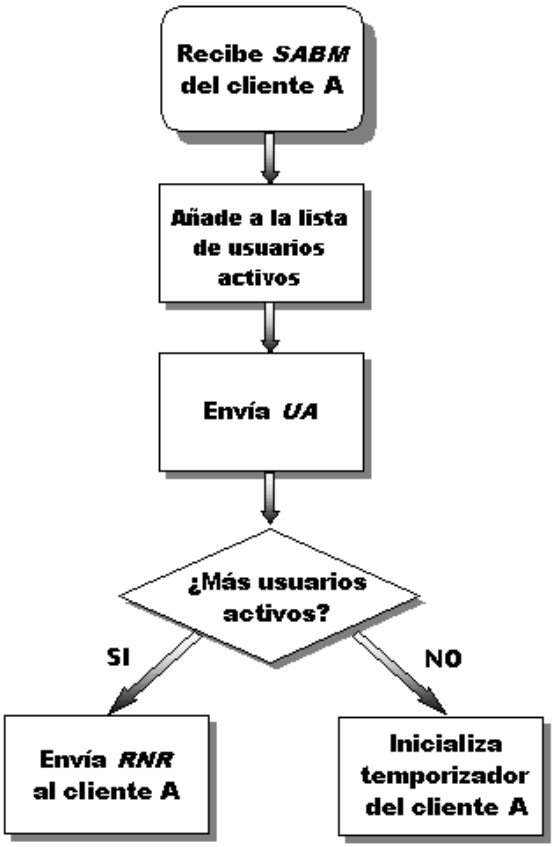


Figura 25: Protocolo DAMA. Establecimiento de conexión

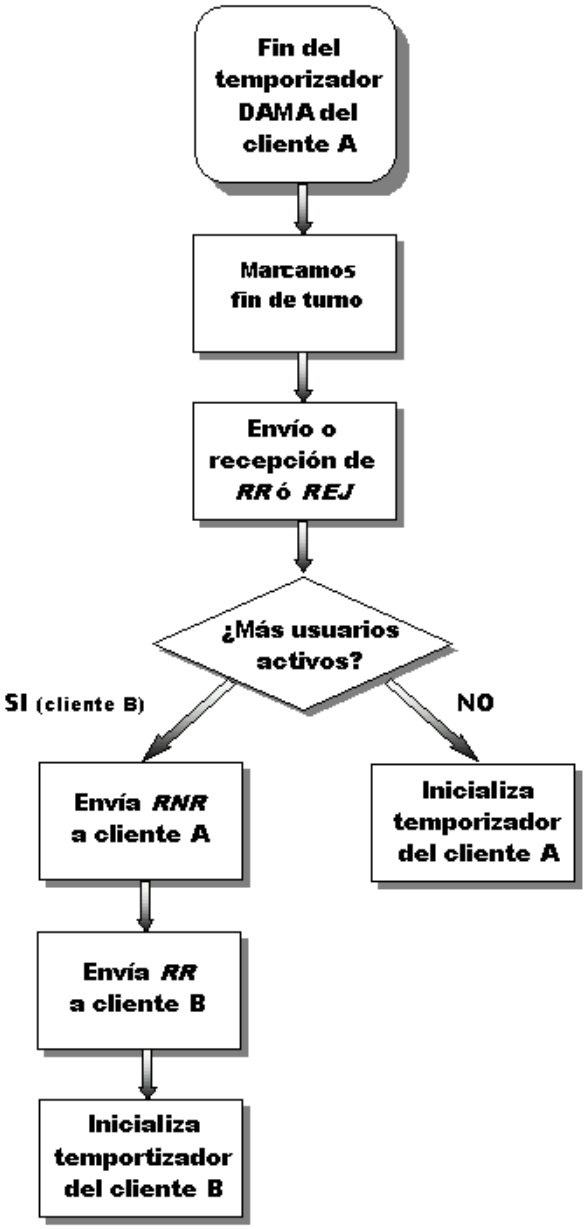


Figura 26: Protocolo DAMA. Fin de turno de un cliente

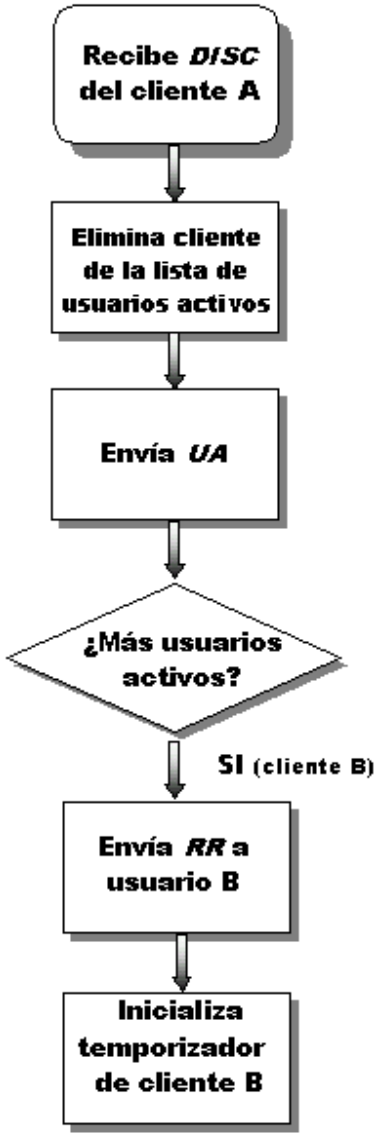


Figura 27: Protocolo DAMA. Petición de desconexión

Antes de entrar en detalle, veamos el cronograma de una comunicación típica entre dos clientes y el servidor. Para simplificar veremos el caso de dos clientes que piden servicio de forma simultánea (figura 28).

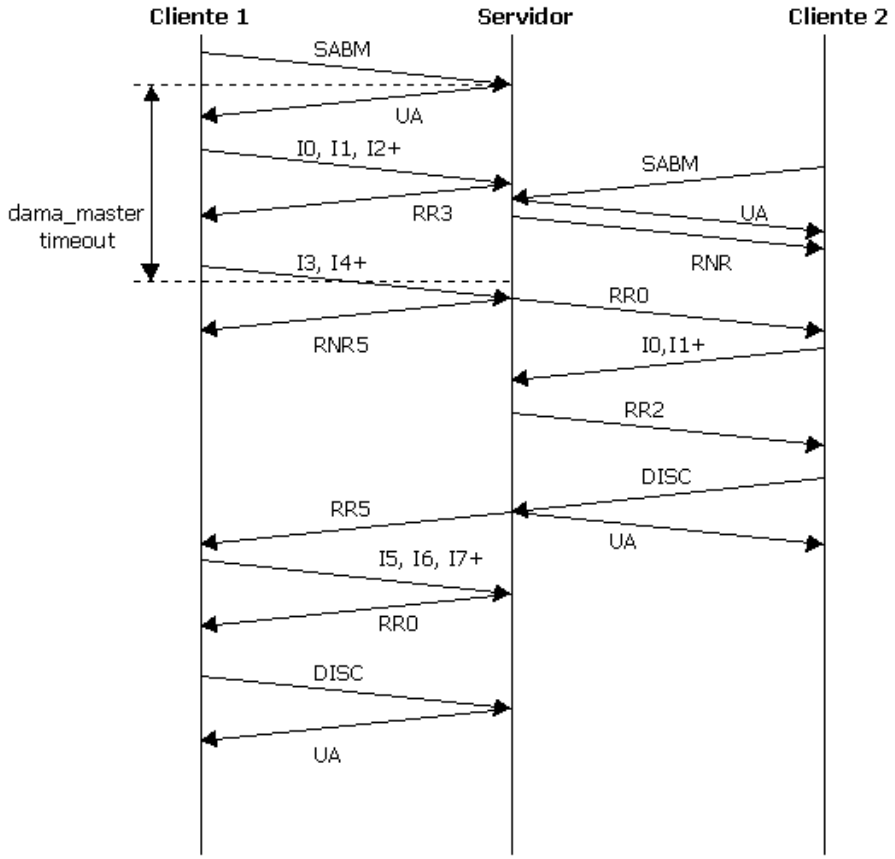


Figura 28: Comunicación de dos clientes con protocolo DAMA

8.3.1. Modificaciones en el kernel

En este apartado veremos en detalle las modificaciones más significativas que han sido necesarias para la implementación de DAMA. Los añadidos se marcan iniciado la línea con el símbolo "+". Los cambios completos se encuentran en el disco que acompaña el proyecto.

include/net/ax25.h Modificaremos alguna estructura y crearemos nuevas constantes:

- Definimos una nueva condición que indica que el servidor tiene una determinada conexión en espera por no tener el turno asignado.

```
 #define AX25_COND_ACK_PENDING       0x01
 #define AX25_COND_REJECT            0x02
 #define AX25_COND_PEER_RX_BUSY      0x04
 #define AX25_COND_OWN_RX_BUSY       0x08
 #define AX25_COND_DAMA_MODE         0x10
+#define AX25_COND_OWN_DAMA_WAIT     0x20
```

- En la estructura ax25_cb, que almacena toda la información referente a una conexión, añadimos los parámetros necesarios:

```
typedef struct ax25_cb {
     struct ax25_cb      *next;
     ax25_address        source_addr, dest_addr;
     ax25_digi           *digipeat;
     ax25_dev            *ax25_dev;
     unsigned char       iamdigi;
     unsigned char       state, modulus, pidincl;
     unsigned short      vs, vr, va;
     unsigned char       condition, backoff;
     unsigned char       n2, n2count;
     struct timer_list   t1timer, t2timer, t3timer, idletimer;
    +struct timer_list   dmtimer;
    +unsigned short      damalost;
    +int                 damachange;
     unsigned long       t1, t2, t3, idle, rtt;
    +unsigned long       dm;
     unsigned short      paclen, fragno, fraglen;
```

```
        struct sk_buff_head write_queue;
        struct sk_buff_head reseq_queue;
        struct sk_buff_head ack_queue;
        struct sk_buff_head frag_queue;
        unsigned char       window;
        struct timer_list   timer;
        struct sock         *sk;        /* Backlink to socket */
    } ax25_cb;
```

include/linux/sysctl.h Añadimos un indicativo adicional para la nueva variable que controla la duración del turno DAMA y que se encontrará en `/proc/sys/net/ax25/[disp]`

```
/* /proc/sys/net/ax25 */
enum {
     NET_AX25_IP_DEFAULT_MODE=1,
     NET_AX25_DEFAULT_MODE=2,
     NET_AX25_BACKOFF_TYPE=3,
     NET_AX25_CONNECT_MODE=4,
     NET_AX25_STANDARD_WINDOW=5,
     NET_AX25_EXTENDED_WINDOW=6,
     NET_AX25_T1_TIMEOUT=7,
     NET_AX25_T2_TIMEOUT=8,
     NET_AX25_T3_TIMEOUT=9,
     NET_AX25_IDLE_TIMEOUT=10,
     NET_AX25_N2=11,
     NET_AX25_PACLEN=12,
     NET_AX25_PROTOCOL=13,
    +NET_AX25_DAMA_MASTER_TIMEOUT=14,
     NET_AX25_DAMA_SLAVE_TIMEOUT=15
};
```

net/ax25 En este directorio hemos añadido código que gestiona la llegada de peticiones de los clientes. Para ello también creamos el nuevo temporizador que gestiona el tiempo de turno.

sysctl_net_ax25.c Añadimos una entrada adicional para la creación del fichero que almacena la variable con el tiempo asignado a cada turno:

```
{NET_AX25_DAMA_MASTER_TIMEOUT, "dama_master_timeout",
 NULL, sizeof(int), 0644, NULL,
 &proc_dointvec_minmax, &sysctl_intvec, NULL,
 &min_dm_timeout, &max_dm_timeout},
```

ax25_timer.c Creamos las funciones que controlan el temporizador del turno DAMA:

```
void ax25_start_dmtimer(ax25_cb *ax25) {
    del_timer(&ax25->dmtimer);
    if (ax25->dm > 0) {
        ax25->dmtimer.data     = (unsigned long)ax25;
        ax25->dmtimer.function = &ax25_dmtimer_expiry;
        ax25->dmtimer.expires  = jiffies + ax25->dm;

        add_timer(&ax25->dmtimer);
    }
}

void ax25_stop_dmtimer(ax25_cb *ax25) {
    del_timer(&ax25->dmtimer);
}

static void ax25_dmtimer_expiry(unsigned long param) {
    ax25_cb *ax25 = (ax25_cb *)param;
    ax25->damachange = 1;
}
```

ax25_in.c Cuando llega una petición de conexión y el modo DAMA está activo, debemos mirar si existen usuarios activos en la lista. Si así es, mandaremos un RNR para silenciar al usuario hasta que el turno de los demás haya finalizado. Si no había ningún otro cliente activo, simplemente activamos el temporizador DAMA del nuevo cliente y la comunicación continúa como siempre.

```
#ifdef CONFIG_AX25_DAMA_MASTER
if (ax25->ax25_dev->values[AX25_VALUES_PROTOCOL]
        == AX25_PROTO_DAMA_MASTER) {
    // Sólo entramos si protocolo DAMA activado

    temp = ax25_list;
        // ax25_list apunta al primer
        // cliente de la lista
    while (temp != NULL) {
        // mientras no encontremos fin de lista
        if(temp->state != AX25_STATE_0) {
            // y se trate de un usario activo
           ax25->condition |= AX25_COND_OWN_DAMA_WAIT;
            // marcamos al nuevo usuario en espera
           ax25_send_control(ax25, AX25_RNR,
                AX25_POLLOFF, AX25_RESPONSE);
            // mandamos RNR al nuevo cliente
           break;
        }
        temp=temp->next;
            // siguiente elemento en la lista
    }
    if(temp == NULL)
        ax25_start_dmtimer(ax25);
        // Si no había más clientes
        // inicializamos el temporizador
        // del turno del cliente

}
#endif
```

ax25_out.c En este fichero se encuentran las función encargadas de mandar los paquetes AX.25 al módem. Si un cliente no tiene el turno, en ningún caso el servidor le debe enviar paquetes de datos. Cuando un cliente recupere el turno, mediante la función **ax25_kick** se servirán todos los paquetes que se hayan retenido.

```
#ifdef CONFIG_AX25_DAMA_MASTER
    if (ax25->condition & AX25_COND_OWN_DAMA_WAIT)
        return;
#endif
```

ax25_std_in.c Si un cliente no disponía del turno DAMA pero aún y así (por pérdida del paquete RNR) manda un paquete de información, se le debe volver a indicar que no dispone del turno. Para ello se añade la comprobación de si ese cliente se encuentra o no en espera en el momento de recibir paquetes:

```
-if (ax25->condition & (AX25_COND_OWN_RX_BUSY) )

// Añadimos la nueva condición
+if (ax25->condition & (AX25_COND_OWN_RX_BUSY |
+    AX25_COND_OWN_DAMA_WAIT))
```

Aunque el tiempo de turno haya finalizado, sólo se hace callar a un cliente cuando éste mande una confirmación de que la ventana ha sido recibida:

```
if (frametype == AX25_RR || frametype == AX25_RNR) {
    if (ax25->damachange == 1) {
        // si el temporizador había llegado a 0
        ax25->damachange = 0;
        // desactivamos el indicativo de cambio
        if(ax25_dm_next(ax25))
        // invocamos la función encargada del
        // cambio de turno.
            return(0);
            // si la función devuelve >0,  había más
            // clientes activos, así que el cliente
            // actual pierde el turno
    }
```

```
}

ax25->damalost=0;
// Reiniciamos el contador de turnos perdidos,
// en caso de llegar a un valor determinado,
// el usuario perderá el turno y será desconectado
```

af_ax25.c Aunque en el uso normal, no es el servidor quien inicia una comunicación con un cliente, se debe contemplar esta posibilidad. Cuando el servidor ha iniciado la petición de conexión se deben hacer las mismas comprobaciones que cuando es el cliente quien ha pedido la conexión:

```
#ifdef CONFIG_AX25_DAMA_MASTER
    if(sk->protinfo.ax25->ax25_dev->
       values[AX25_VALUES_PROTOCOL] ==
                AX25_PROTO_DAMA_MASTER) {
       temp = ax25_list;
       while (temp != NULL) {
           if(temp->state != AX25_STATE_0) {
                sk->protinfo.ax25->condition |=
                        AX25_COND_OWN_DAMA_WAIT;
                ax25_send_control(sk->protinfo.ax25,
                    AX25_RNR, AX25_POLLOFF, AX25_RESPONSE);
                break;
           }
           temp=temp->next;
        }
    if(temp == NULL)
        ax25_start_dmtimer(sk->protinfo.ax25);
    }
#endif
```

ax25_subr.c En este fichero están funciones de uso general, por ello hemos incluido aquí la función *ax25_dm_next* que asigna el turno al siguiente cliente cuando expira el temporizador u otro cliente ha pedido la desconexión:

```
int ax25_dm_next(ax25_cb *ax25) {
    ax25_cb *temp;
    int change;

    temp=ax25->next;
    // temp apunta al siguiente usuario en la lista

    if(temp==NULL)
        temp=ax25_list;
        // Si el siguiente usuario era nulo
        // apuntamos al primer elemento de
        // la lista
    change = 0;
    while(ax25!=temp && temp!=NULL && !change) {
        if(temp->state != AX25_STATE_0) {
            // miramos si el usuario examinado está
            // activo
            if(ax25->state != AX25_STATE_0) {
                // Si el usuario activo también está
                // activo, habrá que indicar RNR

                ax25_send_control(ax25, AX25_RNR,
                        AX25_POLLOFF, AX25_RESPONSE);
                ax25->condition |= AX25_COND_OWN_DAMA_WAIT;
                ax25_stop_heartbeat(ax25);
                ax25_stop_t1timer(ax25);
                ax25_stop_t3timer(ax25);
            }
            // Activamos el nuevo usuario quitando
            // condición DAMA y mandando RR.
            temp->condition &= ~AX25_COND_OWN_DAMA_WAIT;
            ax25_send_control(temp, AX25_RR, AX25_POLLON,
                    AX25_COMMAND);
            ax25_start_dmtimer(temp);
            ax25_start_t3timer(temp);
            ax25_kick(ax25);
            // Los paquetes retenidos ya pueden ser
             enviados
```

```
                change = 1;
                break;
            }
            temp=temp->next;
            // pasamos al siguiente elemento de la lista
            if(temp == NULL)
                temp = ax25_list;
        }
        if(!change && ax25->state!=AX25_STATE_0)
            ax25_start_dmtimer(ax25);
            // Si no habido cambio de turno, reiniciamos el
            // temporizador del cliente actual
        ax25->damalost++;
        if(ax25->damalost==3 && ax25->state!=AX25_STATE_0)
            ax25_std_idletimer_expiry(ax25);
            // Si un cliente ha estado más de dos turnos
            // sin contestar al RR, se cierra la
            // comunicación por inactividad.
        if(change)
            ax25->condition &= ~AX25_COND_ACK_PENDING;
        return(change);
    }
    #endif
```

8.4. Generación e instalación del parche

Para agilizar la modificación de los ficheros necesarios del código del kernel, se crea un fichero en el que únicamente se reflejan los cambios realizados. Con la siguiente utilidad se genera este fichero, denominado **parche**:

Código original: /usr/local/src/linux
Código modificado: /usr/local/src/linux-dama

```
diff -u linux linux-dama -r -B -b -d >dama
```

Al instalar DAMA en el servidor se debe disponer de este fichero para modificar el kernel. Para ello hace uso de la utilidad `patch`:

Directorio actual: /usr/local/src
Código original: /usr/local/src/linux

```
patch -p0 <dama
```

Una vez instalado el parche se tiene que volver a compilar el kernel. Para ello seguimos la secuencia habitual en Debian:

```
make menuconfig
```

En el parche se incluye también una modificación en los menús de instalación. Además de la opción de cliente DAMA, aparecerá una nueva opción de servidor DAMA, que habrá que seleccionar:

```
Code maturity level options
    [*] Prompt for development and/or incomplete code/drivers
Loadable module support
    [*] Kernel module loader
General Setup
    [*] Networking support
    [M] Parallel port support
        [M] PC-Style hardware
Amateur Radio support
    [*] Amateur Radio Support
    [M] Amateur Radio AX.25 Level 2 Protection
        [*] AX.25 DAMA-EHAS Master support
    [M] PicPar
```

Y sólo queda compilar e instalar el kernel:

Debian

```
make-kpkg clean
make-kpkg --revision ax25_1.0 kernel_image
dpkg -i kernel-image-2.2.18_ax25_1.0_i386.deb
```

Aunque se haya instalado el nuevo kernel, la funcionalidad de servidor DAMA no entra en funcionamiento hasta que es activada. Para ello se existe una variable modificable en /proc/sys/net/ax25/[disp]/protocol:

```
0: AX25_PROTO_STD_SIMPLEX
1: AX25_PROTO_STD_DUPLEX
2: AX25_PROTO_DAMA_SLAVE
3: AX25_PROTO_DAMA_MASTER
```

Para que el servidor active el modo DAMA se escribe el valor 3. Si se quiere dejar desactivado escribimos el valor 0. Si se activa el modo DAMA también debe asignarse el tiempo de turno en la variable
`/proc/sys/net/ax25/[disp]/dama_master_timeout`.

8.5. Consideraciones importantes

- La implementación realizada se ha hecho de tal forma que nunca se quite el turno a un cliente (cualquiera que sea la dirección de transmisión en ese momento) hasta que la ventana en curso ha sido totalmente transmitida. Ya hemos visto que la eficiencia del protocolo AX.25 mejora si se utilizan el tamaño máximo de ventana, así que era importante hacerlo así.

  Así que uando el tiempo de turno DAMA finaliza (*damachange = 1*) se espera a la recepción o envío de un paquete RR, y en ese momento se produce el cambio de turno.

- Respecto al tiempo de turno de los clientes, éste debe ser lo suficientemente grande para que la sobrecarga que implica DAMA no sea apreciable, pero no tan grande como para que que los usuarios aprecien una inactividad excesiva en sus terminales. Las pruebas que se han realizado se han hecho con un tiempo de 10 segundos con resultados satisfactorios.

- Las colisiones en un sistema DAMA aún son posibles, veamos en qué casos se producen:

  1. En la fase de conexión se producirá colisión si el cliente que la efectúa no escucha a otro cliente que esté transmitiendo en ese mismo momento. Las posibilidades de que se produzca esta colisión aumenta si la comunicación que en ese momento está activa en un acceso a SMTP

(envío de correo), pues los paquetes que el cliente activo está enviando al servidor no son escuchados por el usuario que manda la petición de conexión.

2. Si se pierde el paquete RNR que indica el fin de transmisión de un cliente, este continuará enviando paquete hasta que el RNR sea recibido. La posibilidad de una colisión por esta causa depende, a diferencia de la anterior, de la tasa de error del canal.

8.6. Aplicaciones auxiliares

En el caso que tengamos un esquema de servidor DAMA dedicado (figura 29), aun faltará que todos los mensajes almacenados en el servidor DAMA pasen al servidor del Centro de Salud para ser dirigidos posteriormente a Internet. Para ello, el servidor Linux del Centro de Salud se deberá conectar como un cliente más al servidor DAMA y a través de una conexión UUCP traer el correo almacenado.

Dado que el servidor del Centro de Salud tiene el sistema operativo Linux, el *daemon* cliente para Windows no nos sirve. Por ello se ha realizado, con la misma idea, un programa (`sshax25cliente`) que escucha en el puerto de SSH y crea la conexión AX.25 con el servidor DAMA cuando es necesario, sirviendo luego de pasarela de datos entre ambos puertos.

Para invocar el programa se hace lo siguiente:

```
sshax25cliente puerto callsign_origen callsign_destino
```

Donde:

puerto. Nombre del puerto del enlace radio definido en `/etc/axports`.

callsign_origen. Indicativo del ordenador en el centro de Salud.

callsign_destino. Indicativo del servidor DAMA.

La llamada a este programa se puede incluir en el *script* de arranque (ver Anexo J). Este programa será útil únicamente en el servidor del Centro de Salud y siempre que éste no sea el que realiza las funciones de servidor DAMA.

Este *daemon* tiene la particularidad que en vez de escuchar del puerto SSH estándar (22), lo hace de uno alternativo (22000), porque el primero de los puertos está ya en uso por el servidor SSH instalado. Por ello, cuando queramos crear el túnel SSH con el servidor DAMA, deberemos llamar al cliente SSH con un parámetro especial:

```
ssh -p 22000 usuario@localhost
```

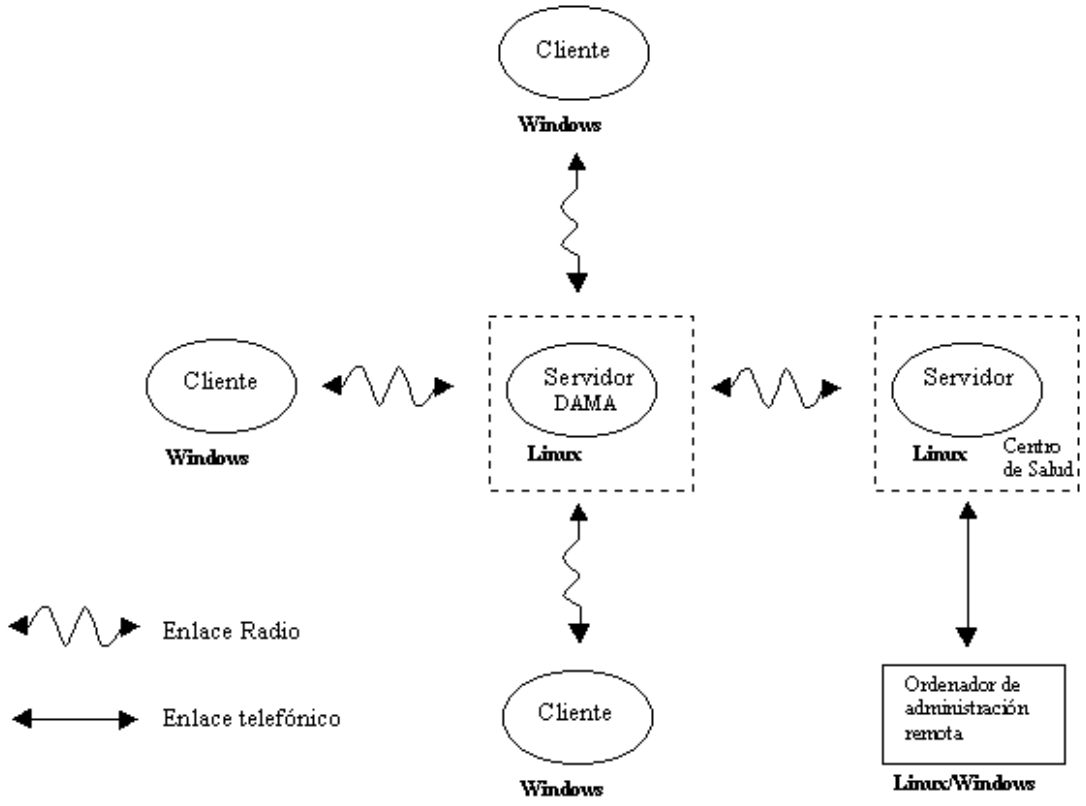


Figura 29: Arquitectura de red con servidor DAMA dedicado

8.7. Rendimiento de DAMA

Si se pone un tiempo de turno mínimo (aprox. 10 segundos), los tiempos de commutación de una estación a otra en los cambios de turno serán despreciables, por lo que el rendimiento que ve el usuario es:

$$v_{usuario} = \frac{v_{max}}{n}$$

Donde:

v_{max} = Velocidad con un único cliente

n = Número de clientes activos

En cualquier caso, también deberían tenerse en cuenta que se pueden producir colisiones en los casos contemplados anteriormente, por lo que sería muy difícil hacer un estudio teórico exacto. Por ello en próximos capítulos sé harán simulaciones de rendimiento real con varios clientes.

9. Gestión remota

Los usuarios finales del sistema son personal médico con conocimientos informáticos limitados, por ello se creyó interesante dotar a la red de un sistema de gestión remota posible desde cualquier ordenador conectado a Internet.

El objetivo es no sólo permitir la administración del servidor DAMA o del servidor del centro de salud, lo cual es sencillo por tratarse de máquinas Linux, sino también de los clientes Windows. Esto último plantea la dificultad de que Windows no incorpora ningún método de gestión remota, ni siquiera el acceso a un intérprete de órdenes para realizar operaciones básicas.

De tal forma, se pensó que el mismo *daemon* que se encargaba de interceptar la conexión a los puertos de SMTP y POP3, podía servir para recibir peticiones de conexión AX.25 del servidor e iniciar la ejecución de un intérprete de órdenes, redirigiendo la salida y entrada estándar al canal radio.

Otra dificultad que habrá que solventar es que no conocemos la dirección IP del servidor del Centro de Salud que se conecta a Internet por lo que habrá que diseñar un modo de saberla.

9.1. Esquema del sistema de gestión remota

Como ya vimos en la Introducción, podemos tener dos escenarios dependiendo si el servidor DAMA es el propio ordenador del Centro de Salud o no:

- Servidor DAMA en el centro de Salud. Figura 30
- Servidor DAMA dedicado. Figura 31

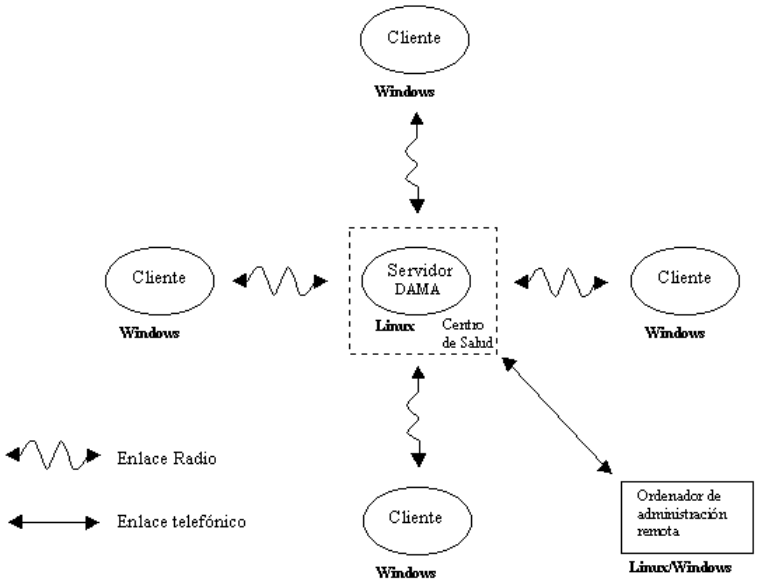


Figura 30: Arquitectura de red. Servidor DAMA en el Centro de Salud

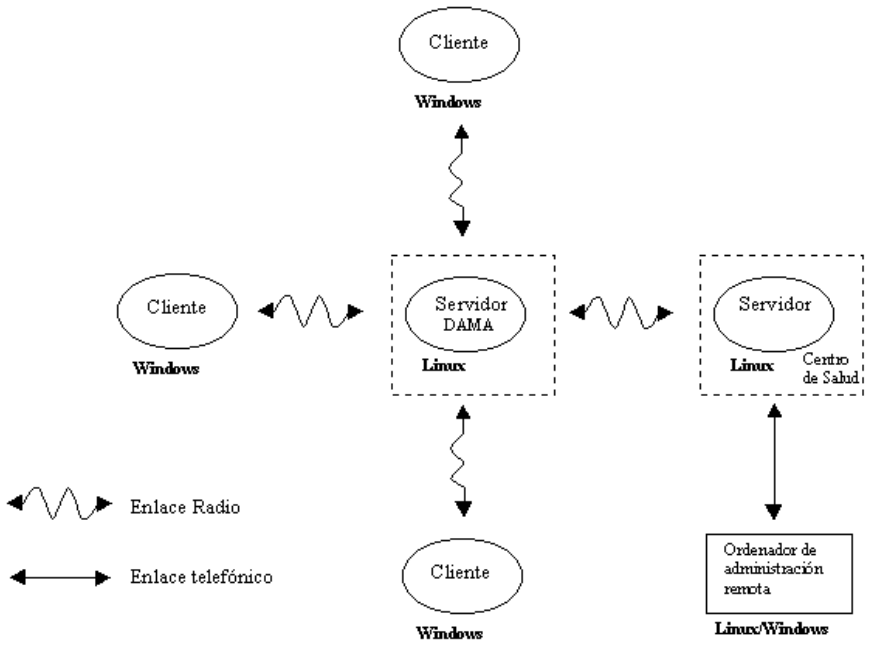


Figura 31: Arquitectura de red con servidor DAMA dedicado

9.2. Gestión del cliente Windows

Cuando un cliente recibe una petición de conexión externa, inicia la subrutina *shell*. Lo primero que hace es pedir una contraseña para entrar (que se encuentra en el fichero de configuración (`daemon.cfg`). Si la autentificación es correcta, se abre un proceso del `COMMAND.COM` (intérprete de órdenes) y se redireccionan la entrada estándar (*stdin*) y la salida estándar (*stdout*) a dos *handles* (identificadores de canal) a los que tenemos acceso y de los que podemos leer y escribir.

Es importante destacar que el cliente no podrá ni escribir ni leer correo durante el proceso de gestión remota. Durante este proceso el *daemon* no está escuchando el puerto SSH así que cualquier intento de lectura o escritura de mail dará un error en el cliente de correo.

Con este sistema se permite realizar las operaciones que se pueden hacer normalmente en una *shell* de Windows. Se consideró que en algunos casos podía ser interesante realizar una transferencia de ficheros entre cliente y servidor, y se estableció un sistema simple y sencillo para ello. Si la orden que se entra en el prompt empieza por algún símbolo del tipo ">"ó "<", se intercepta y se pasa a modo de transferencia de fichero. Veamos la sintaxis en detalle:

>**nombre_fichero.** Envía un fichero desde el directorio actual del servidor Linux al directorio donde está instalado el *daemon* en Windows.

<**nombre_fichero.** Envía un fichero desde el directorio de instalación del *daemon* de Windows al directorio actual del servidor Linux.

Estos caracteres son introducidos en el programa **wtelnet** que realiza la conexión desde el servidor. Dicho programa se analiza con profundidad más adelante en este mismo capítulo.

9.2.1. Implementación en el cliente

Si recordamos el pseudocódigo que describe el comportamiento del *daemon* cliente del apartado 6, veremos que cuando se produce una petición de conexión desde el servidor se inicia la función *shell*. Veamos el pseudocódigo de esta función:

```
Petición de contraseña;
si contraseña no correcta
        retorna de la función;
crea canal stdin;
crea canal stdout;
arranca intérprete de órdenes;
mientras COMMAND.COM activo
        si datos en canal stdout
                lee datos de stdout;
                escribe datos en canal AX.25;
        si datos en canal AX.25
                lee datos en AX.25;
                escribe datos en stdin;
        si recibe >
                lee nombre fichero;
                lee n = bytes_a_escribir;
                crea fichero para escritura;
                escribe fichero;
                si transferencia correcta
                        manda "EOT" (End of transmission);
                sino manda "ERR" (Error);
        si recibe <
                lee nombre fichero;
                abre fichero para lectura;
                manda n = bytes_a_transmitir;
                escribe fichero en canal AX.25;
        si recibe DISC (desconexión remota)
                sale del bucle;
si recibido DISC
        cierra intérprete de órdenes;
sino manda DISC;
cierra canales stdin, stdout;
```

9.2.2. Implementación en el servidor

En el lado del servidor deberemos diferenciar el caso en que el servidor DAMA esté en el Centro de Salud o no lo esté. Esto es así porque será siempre el Centro de Salud el que tenga conexión telefónica y por tanto posibilidad de acceso a Internet.

El programa `wtelnet` debe interceptar la secuencia CONTROL + C, pues esta secuencia por defecto finaliza la ejecución del programa y lo que desearíamos es que tuviera efecto en la *shell* de Windows. Para ello se asigna la señal de interrupción (*signal*) SIGINT a la función `controlc_handler`. Esta función envía el caracter `0x3` que en MSDOS se interpreta como la secuencia CONTROL+C.

Por otra parte el programa envía el señal SIGUSR1 al proceso `sshax25d` al iniciar la gestión y envía el SIGUSR2 al finalizar. El proceso `sshax25d` recibe las señales y las interpreta de la siguiente forma:

- SIGUSR1: Bloquea la entrada de nuevos clientes en el *daemon* Linux. De esta forma el programa de gestión remota (que ya está activo en el *daemon*) tiene acceso exclusivo al enlace radio y puede realizar la tarea de administración de forma más cómoda y rápida. En los clientes de correo se recibe una desconexión inmediata y muestran por pantalla el correspondiente mensaje de error. Así pues la administración debe ser lo más rápida posible para no interferir el acceso al correo de los otros clientes.
- SIGUSR2: Una vez finalizada la administración remota ya se puede volver a permitir la entrada de nuevos clientes. Cuando el *daemon* sshax25d recibe esta señal vuelve a permitir la entrada.

La llamada al programa es la siguiente:

Uso: `wtelnet puerto callsign_origen callsign_destino`

Las operaciones que realiza el programa `wtelnet` son las siguientes:

```
conecta con indicativo_destino;
pide pasword al usuario;
envía contraseña al daemon cliente;
si contraseña no correcta
        sale del programa;
envía SIGUSR1 a sshax25d;
mientras stdin y socket AX.25 abiertos
        si datos en stdin
                lee datos stdin;
                si empieza por >
                        abrimos fichero para lectura;
                        mandamos nombre de fichero;
                        mandamos tamaño de fichero;
                        mandamos contenido del fichero;
                        esperamos contestacion del cliente;
                        si recibido "EOT", todo correcto;
                        si recibido "ERR", reportar error;
                si empieza por <
                        mandamos cadena de peticion;
                        recibimos el tamaño del fichero;
                        abrimos el fichero para escritura;
                        escribimos el contenido del fichero;
                sino escribe datos en socket AX.25;
        si datos en socket AX.25
                lee datos de socket AX.25;
                escribe datos en stdout;
envía SIGUSR2 a sshax25d;
```

9.2.3. Servidor DAMA en el Centro de Salud

Este es el caso más simple. Hasta el Centro de Salud se puede entrar con un cliente `ssh` de la forma habitual. Para poder entrar después en el cliente Windows, se utiliza el programa `wtelnet`, descrito en el apartado anterior.

9.2.4. Servidor DAMA dedicado

En esta configuración es necesario un salto adicional (ver figura 31) que en el caso anterior. Veamos la secuencia:

1. Desde el ordenador en Internet entramos en el servidor del Centro de Salud mediante un conexión SSH.

2. Desde el servidor del Centro de Salud hay que entrar en el servidor DAMA. Para ello se ejectua el programa residente `sshax25cliente`. Este programa intercepta las conexiones de los clientes SSH en el puerto 22000. No se puede usar el puerto 22 porque este ya está siendo usado como servidor SSH. Para activar el `sshax25cliente` hay que seguir la siguiente sintaxis:

   ```
   sshax25cliente puerto indicativo_serv indicativo_serv_dama
   ```

   Este *daemon* puede quedar siempre activado porque no interfiere el funcionamiento normal del servidor del Centro de Salud. Así pues, se puede añadir su ejecución al *script* de arranque correspondiente a dicho servidor.

   Una vez el *daemon* está en memoria podemos realizar una conexión a él:

   ```
   ssh usuario@localhost -p 22000
   ```

   Vemos que hay que decirle explícitamente que el puerto a conectarse es el 22000, en caso contrario lo haría en el puerto 22 donde no haría más que entrar en la propia máquina.

   El *daemon* se encarga de dirigir la petición del cliente SSH al canal radio. Nos identificamos ante el servidor DAMA y una vez ya hemos entrado en él, nos encontramos en la misma situación que el apartado anterior: sólo queda usar `wtelnet` para entrar en el cliente Windows.

Si nos fijamos de nuevo en la figura 31 veremos que estamos usando simultáneamente dos enlaces radio:

- Servidor Centro de Salud $\longleftrightarrow$ Servidor DAMA
- Servidor DAMA $\longleftrightarrow$ Cliente Windows.

Se hace imprescindible, pues, deshabilitar el protocolo DAMA. Si no lo hiciéramos, el servidor DAMA otorgaría turnos de forma secuencial a estos dos enlaces y cualquier petición tardaría un mínimo del tiempo de un turno en tener efecto, lo que haría el proceso de gestión realmente tedioso.

Cabe resaltar que el hecho de desactivar el protocolo DAMA en este caso concreto (administración remota) no afecta de forma negativa a la comuncación ni provoca colisiones de paquetes. Si se analiza el tráfico en una administración se ve que un proceso habitual de **petición-respuesta** no provoca transmisiones simultáneas de clientes a servidor DAMA, que es el caso en que las colisiones serían inevitables.

Así, además de gestionar el envío de las señales SIGUSR1 y SIGUSR2, el programa **wtelnet** se encarga de desactivar el protocolo DAMA, para ello se escribe **0** en /proc/sys/net/ax25/[disp]/protocol. Acabada la gestión es imprescindible volver a activar DAMA para que la red vuelva al estado normal (escribe 3 en /proc/sys/net/ax25/[disp]/protocol). Para más detalle de la funcionalidad de estos ficheros ver Anexo C.

9.3. Gestión del servidor Linux desde un cliente Windows

Hasta ahora sólo nos hemos centrado en la gestión que desde el servidor Linux se puede hacer en los clientes Windows. También se puede hacer una gestión remota del servidor Linux desde los clientes Windows, mediante el uso directo del programa SSH32. Para ello se ejecuta directamente esta aplicación (sin activar ningún cliente de correo) y el *daemon* se encargará de redirigir la conexión por el enlace radio y se abrirá una *shell* en el servidor.

Si nos interesara hacer transferencias entre el cliente y el servidor también podemos usar el programa `SCP` (secure copy) incluído en el paquete del *daemon* de Windows. El `scp` tiene la siguiente sintaxis:

```
scp.exe origen destino
```

El parámetro que sea un fichero remoto tendrá la siguiente sintaxis:

```
usuario@host:/[ruta]/fichero
```

Así pues, si quisieramos copiar el fichero `config.sys` al directorio `home` del usuario `prueba` haríamos:

```
scp c:\config.sys prueba@localhost:/home/prueba/config.sys
```

Notar que como siempre el *host* a poner en la orden es *localhost*, de esta forma el **scp** conecta con el puerto 22 y el *daemon* de Windows empieza su función de pasarela. En estos casos de administración los *hebras* de SMTP y POP3 no intervienen para nada.

9.4. Alcance de la gestión remota

Vamos a hacer un repaso de las posibilidades que nos permite el sistema de gestión remota descrito en el presente apartado:

- Copiado, borrado, renombrado de ficheros y en general cualquier operación realizable desde la línea de órdenes MsDos.
- Transferencia de ficheros entre el servidor y el cliente (mediante los caracteres > y <)
- Edición de ficheros de texto. No es posible el uso de programa de pantalla completa para la edición de ficheros. Por ello en el paquete se incluye un antiguo programa de edición de línea presente hasta la versión 3.0 de MsDos (ver Anexo H)
- Edición del registro. El programa REGEDIT permite la exportación e importación de claves. Las opciones disponibles son las siguientes:

  regedit /e registro.txt Exporta el registro entero a un fichero de texto.

  regedit /e registro.txt <clave de registro> Exporta una clave o un árbol de claves del registro a un fichero de texo . Permite una fácil edición remota (con EDLIN) pues el fichero generado es pequeño. Ej:

  regedit registro.txt. Actualiza el registro de Windows con los cambios que haya en el fichero `registro.txt`, tanto si éste contiene el registro entero o sólo una parte.

  Para más detalles consultar el Anexo I.

- Restauración automática de imágenes mediante el programa PQDI de PowerQuest (ver próximo apartado y Anexo G)

9.4.1. Restauración automática de imágenes

Como la gestión remota tiene sus limitaciones y se puede dar la posibilidad de una corrupción general del sistema Windows o sus aplicaciones, se ha implementado un sistema que permite la restauración del sistema operativo y de las aplicaciones instaladas de forma remota.

Estructura de particiones de los clientes Windows

Para poder hacer una restauración selectiva de particiones se ha establecido la siguiente partición del disco duro:

Partición 1. Unidad C: Sistema Operativo y Aplicaciones Windows.

Partición 2. Unidad E: Documentos de usuario.

Partición 3. Unidad F: Imágenes para restaurar.

De esta forma podremos restaurar solamente las particiones que interesen sin destruir datos de usuario.

Scripts de restauración automática

El programa PQDI de PowerQuest incluido en la aplicación Partition Magic [17] incorpora un control gráfico mediante ventanas para la creación y restauración de imágenes. La gestión remota no permitiría la ejecución e interacción con un sistema de estas características, pero PQDI también incluye un sistema automático mediante el uso de *scripts*. En este punto se presenta el problema que, como es obvio, la restauración de las particiones sólo se puede realizar en modo MsDos puro. Dado que el *daemon* que permite la gestión remota sólo está activo en Windows, hay que implementar algún método para ello.

El método escogido ha sido la sustitución del fichero de arranque (`autoexec.bat`). Cuando el ordenador rearranque y ejecute el nuevo `autoexec.bat`, se ejecutarán las órdenes necesarias para la restauración del sistema completo.

El *script* de restauración sería el siguiente:

```
SELECT DRIVE 1
DELETE PARTITION 1
SELECT FREESPACE FIRST
SELECT IMAGE ALL
RESTORE
REBOOT
```

y en el nuevo `autoexec.bat` incluiremos esta línea:

```
G:\PQDI /CMD=G:\SCRIPT1.TXT /IMG=G:\IMAGEN.PQI
```

Así pues, la secuencia completa para la restauración de imágenes sería la siguiente:

1. Entrar en el cliente Windows de la forma descrita anteriormente.
2. Incluir la nueva línea en el `autoexec.bat`. Para ello se puede usar el programa EDLIN o substituir el `autoexec.bat` por uno alternativo que incluya la mencionada línea.
3. Rearrancar el cliente. Para ello se hace uso de la biblioteca `SHELL32.DLL` de Windows. Dentro de esta biblioteca encontramos la función *SHExitWindowsEX* con los siguientes parámetro:

   1: LOGOFF (Sale de la sesión actual)
   2: REBOOT (Reinicia el ordenador)
   4: FORCE (Fuerza la finalización de los programas activos)
   8: SHUTDOWN (Apaga el ordenador)

   Los parámetros pueden superponerse: así, nuestro programa necesitaría:

   1: LOGOFF
   2: REBOOT
   4: FORCE

   Párametro $= 1 + 2 + 4 = 5$

Así pues, la orden a ejecutar sería:

```
RUNDLL32.EXE shell32.dll,SHExitWindowsEx 5
```

4. En este momento el ordenador se reiniciará y perderemos la conexión establecida con él. Después del tiempo estimado de restauración se podrá volver a conectar con el cliente y se podrá comprobar si la estructura del disco duro es la correcta.

9.5. Determinación IP del Centro de Salud

Hasta ahora no nos hemos preocupado de un aspecto importante: desconocemos la dirección IP del primer ordenador en el que hay que entrar en el proceso de gestión remota, que es el servidor Linux del Centro de Salud (que puede ser a su vez servidor DAMA). Este ordenador, que no dispone de conexión permanente a Internet, se conecta de forma periódica para llevar y traer el correo hacia el exterior.

De lo que si disponemos, como es lógico, es del número de teléfono del Centro de Salud al que está conectado el ordenador a través de un módem. Con una llamada a través de otro módem, podriamos entrar en el servidor, pero entonces se trataría de una llamada nacional o internacional y tendría un coste muy alto. Por ello se ha diseñado un sistema con el que indicamos al servidor que se debe conectar a Internet (con lo que ya se trataría de llamada local) y que debe informar de su dirección IP (que será asignada dinámicamente en cada conexión por el protocolo PPP).

Para ello creamos una cuenta especial en el servidor con la siguiente entrada en el fichero `/etc/passwd`:

```
llamame:x:1003:1003:,,,:/home/llamame:/usr/local/sbin/rellamada
```

El último parámetro de esta entrada representa la *shell* que usa y por tanto será el primer programa a ejecutar cuando se produzca un *login* en este usuario.

En el momento que queramos entrar en esta cuenta aún no dispondremos de la dirección IP, por ello habrá que entrar haciendo una llamada con un módem:

```
cu -l /dev/ttyS2
atdt #telefono
login: rellamada
password: ******
```

El *script* `/usr/local/sbin/rellamada` tiene la siguiente forma:

```
#!/bin/sh
PATH=/bin:/usr/bin
cd /tmp
echo "" echo "ahora llamo"
echo "" at now + 1 minutes  << FIN
  wget 'http://central.ehas.org/rellamada.php?‘hostname‘'
    > /dev/null 2>&1
FIN
exit 0
```

El programa `wget` es una aplicación de Linux que realiza un acceso a la página web especficada. En este caso accede a un servidor propio y le envía como parámetro el nombre del *host* local. El fichero `rellamada.php` que se encuentra en **central.ehas.org** tiene el siguiente contenido:

```
<?php
   $a=fopen("/tmp/rellamada","a+");
   fputs($a,date("d/m/Y h:i:s > ", time()).$QUERY_STRING."
        llama desde ".$REMOTE_ADDR."\n");
   fclose($a);
?>
```

La llamada a `wget` tiene éxito porque en el servidor del Centro de Salud está instalado el programa `diald`, encargado de interceptar una petición de acceso a Internet y realizar la llamada al proveedor de acceso configurado.

Una vez realizado el proceso completo, la dirección IP del ordenador que realizó la petición (que será el ordenador del Centro de Salud) quedará almacenada en el fichero `/tmp/rellamada` de una máquina permanentemente conectada a Internet (central.ehas.org) donde la podremos consultar.

10. Estudio de prestaciones

Las pruebas realizadas en este apartado se han llevado a cabo con el siguiente equipamento:

- Ordenadores portátiles Airis H-2122 a 200 MHz. 16Mb RAM.
- Módems PicPar.
- Para la transmisión a 1200bps se han usado radios Motorola PRO3100. Para 9600bps las radios son Yaesu VX-2000.
- En vez de antenas (que podrían interferir en otros usuarios), se han usado cargas que permiten la comunicación entre las radios sin interferir en el espacio radioeléctrico.

10.1. Comparativa a 1200bps

En este apartado veremos la mejora de rendimiento que experimenta el sistema con la adopción del nuevo método propuesto en el presente proyecto.

Para el análisis de velocidad se han tomado mensajes cuyo contenido son archivos HTML de los que se están usando para cursos médicos reales. Esto se hace así porque serán los ficheros más habituales y de mayor tamaño que se envíen, y como la tasa de compresión del canal SSH depende del tipo de información, es interessante hacer las pruebas con ficheros reales.

Las pruebas se han hecho en accesos a POP3 (lectura de correo). Los tiempos para accesos a SMTP (escritura de correo) serán prácticamente idénticos. En la próxima página podemos ver los resultados obtenidos con el sitema antiguo (**Ethrax25**) en una lectura de correo.

Fichero envíado (Kbytes)	24	50	80
Autentificación POP3 (seg)	25	25	25
Transferencia del mensaje (seg)	482	1020	1580
Total (seg)	507	1045	1605
Velocidad (bps) sin autentificación	398	392	405
Velocidad (bps con autentificación	378	382	398

Figura 32: Rendimiento del sistema antiguo (Ethrax25) a 1200bps

Si observamos la velocidad de transferencia sin contar la autentificación vemos que es de aproximadamente 400 bits/seg = **50 bytes/seg**. Si añadimos ahora la fase de autentificación, el tiempo que tardará en recibir (o enviar) un mensaje en Ethrax25 sería (en ausencia de errores de transmisión):

$$t_{transferencia}(segundos) = \frac{tamanyo\_mensaje(octetos)}{50} + 25$$

Veamos ahora las prestaciones obtenidas con el nuevo sistema basado en SSH. En este caso habrá que definir unos parámetros que en el caso anterior no existían:

Octetos enviados efectivos. Son el número de octetos que ocupa el mensaje que se va a transmitir.

Octetos enviados canal radio. Gracias a la compresión del SSH, el número de octetos enviados por el canal radio será menor. En este campo se indicará el número de octetos que realmente se han enviado por dicho canal sin contar la autentificación.

Tasa de compresión. Cociente entre los dos parámetros anteriores.

Autentificación SSH. Es el tiempo que tarda el cliente en autentificarse mediante el sistema RSA. Finalizado este tiempo el túnel seguro SSH está creado.

Autentificación POP3. De la forma habitual, el cliente de correo deberá autentificarse ante el puerto a la hora de leer el correo.

Velocidad del enlace. Aquí se indica la velocidad media durante la fase de transferencia del mensaje.

Octetos enviados efectivos (Kbytes)	24	50	80
Octetos enviados canal radio (Kbytes)	8.1	15.3	24.3
Tasa de compresión	2.96	3.26	3.29
Autentificación SSH (seg)	25	25	25
Autentificación POP3 (seg)	25	25	25
Transferencia del mensaje (seg)	60	118	192
Total (seg)	110	168	192
Velocidad del enlace (bps)	1083	1037	1012
Velocidad efectiva de transferencia (bps)	3200	3389	3333
Velocidad efectiva con autentificación (bps)	1745	2380	2644

Figura 33: Rendimiento del nuevo sistema a 1200bps. 24, 50 y 80 Kbytes.

Velocidad efectiva de transferencia. Independientemente de la velocidad del enlace, la velocidad que el usuario aprecia en la transferencia del correo es mayor, dependiendo de la tasa de compresión alcanzada.

Velocidad efectiva. Este parámetro es idéntico al anterior con la diferencia que sí se tiene en cuenta el tiempo de autentificación (tanto del SSH como del POP3).

Si representamos el número de octetos transmitidos en función del tiempo en ambos sistemas obtenemos la siguiente gráfica:

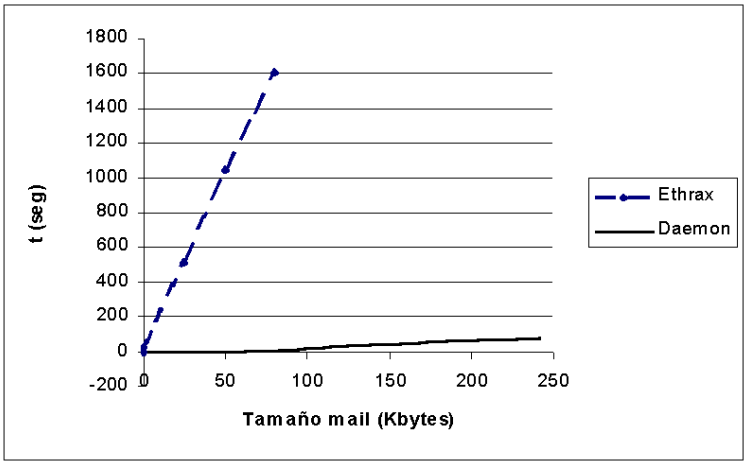


Figura 34: Comparación de prestaciones a 1200bps

Únicamente para mensajes de tamaño muy pequeño el sistema antiguo es mejor que el nuevo (por ahorrarse el tiempo de autentificación SSH). Para saber con que tamaño mínimo el nuevo sistema supera al anterior en prestaciones, realizamos unas pruebas expermientales con mensajes de 1, 1.5 y 2Kbyes:

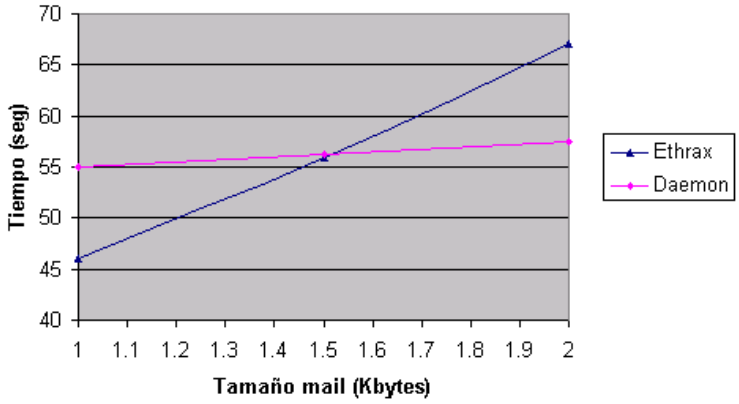


Figura 35: Comparación de prestaciones a 1200bps para mensajes cortos

Aproximadamente para mensajes mayores de 1.5 Kbytes el nuevo sistema es más rápido que el anterior. Si tenemos en cuenta que sólo la cabecera del mensaje ya ocupa 1Kbyte, el resultado obtenido es satisfactorio.

10.2. Estudio de prestaciones a 9600bps

En el apartado anterior hemos visto que la compresión que se consigue depende del tamaño del fichero que enviemos. El parámetro que sí se mantiene más o menos constante es, como era de esperar, el que habíamos denominado **velocidad del enlace**. Veamos el resultado obtenido en el envío de un fichero de 100Kbytes:

Octetos enviados efectivos	100 Kbytes
Octetos enviados canal radio	31 Kbytes
Tasa de compresión	3.22
Autentificación SSH	12 seg
Autentificación POP3	15 seg
Transferencia del mensaje	44 seg
Total	71 seg
Velocidad del enlace	5636 bits/seg
Velocidad efectiva de transferencia	18181 bits/seg
Velocidad efectiva (con autentificación)	11267 bits/seg

Es necesario destacar algunos puntos importantes en comparación con las prestaciones a 1200bps:

- La velocidad del enlace no es tan elevada, en proporción, que en el estudio anterior: recordemos que entonces obteniamos unos 1067 bps en un enlace 1200 mientras que aquí obtenemos algo más de 5600 bps en un enlace 9600. Esto se debe a que cuanto mayor es la velocidad del enlace más empiezan a tomar peso los valores de *txdelay* y *txtail*. Por mucho que aumentemos la velocidad del enlace la velocidad se vería limitado por estos dos valores.

- El tiempo de autentificación, aunque se ha reducido, continúa siendo elevado y no ha experimentado una mejora tan apreciable como el de la velocidad efectiva de transferencia. Esto se debe principalmente a que en la fase de autentificación el cliente y servidor están continuamente enviando y recibiendo paquetes por lo que son nuevamente los parámetros de *txdelay* y *txtail* los que limitan la velocidad (por el tiempo de commutación entre transmisión y recepción). Además, el tamaño de los paquetes es tan reducido que la compresión SSH es totalmente ineficaz.

En el apartado anterior se vio el rendimiento del sistema para tres tamaños de mensajes diferentes. En este caso haremos un estudio más exhaustivo, cubriendo mensajes de 10 a 300 Kbytes. En las pruebas que se han realizado se ha visto que los tiempos de autentificación eran siempre iguales, al igual que los de **transferencia del enlace**. Como el único factor variable es el de la tasa de compresión en función del tamaño del mensaje, nos centraremos en este aspecto. Veamos la gráfica obtenida:

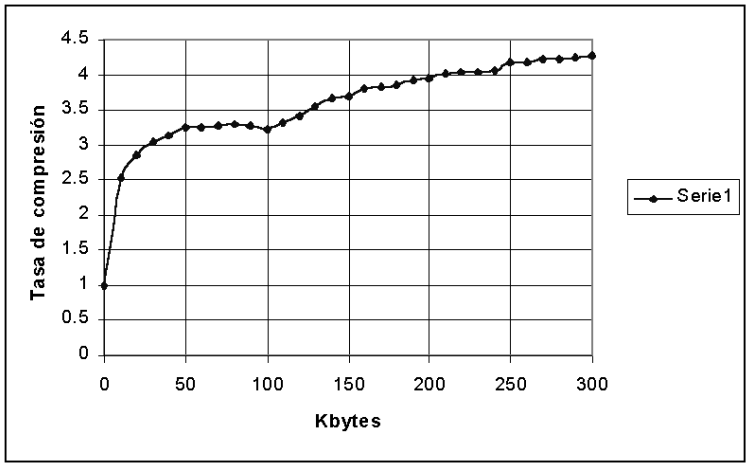


Figura 36: Tasa de compresión del SSH

Si dejamos el valor de la tasa de compresión en función de esta gráfica, en tiempo de transmisión de un mensaje será:

velocidad efectiva = 5500 bits/seg = 687 octetos/seg

$$t_{transferencia}(segundos) = 12 + 15 + \frac{tamanyo\_mensaje(octetos)}{tasa\_de\_compresion(tamanyo)*687}$$

10.3. Estudio de prestaciones de DAMA

Se han realizado pruebas hasta con tres clientes simultáneos y las velocidades de transferencia alcanzadas son parecidas a los alcanzados en el apartado anterior con la salvedad de que la velocidad que aprecia un cliente es menor en función del número de clientes activos. Las pruebas se han realizado de la siguiente forma:

1. Se realizan transferencias de mensajes de 100Kbytes. Estos mensajes se envían del cliente al servidor (acceso a SMTP), que es el caso en que el número de colisiones puede ser mayor (ver sección 8.5).

2. El orden de peticiones de conexión de cada cliente al servidor coincide con el subíndice asignado al tiempo.

3. Las pruebas se han hecho de tal forma que la entrada de nuevos clientes se produzca justo en el momento en que el cliente anterior ha finalizado la fase de autentificación. Esto se ha hecho así porque en una situación real lo habitual será que un cliente pida conexión una vez los demás están en la fase de transferencia de mensajes.

4. El resultado que se buscar es el de **sobrecarga**, parámetro que entenderemos como el porcentaje adicional que tarda la escritura del correo respecto al caso en que se produjera de forma secuencial (sólo un cliente simultáneo).

1 Cliente

$t_{autentificacion} = 28seg$
$t_{transmision} = 44seg$
$t_{total} = 72seg$

2 Clientes

$t1_{autentificacion} = 28seg$
$t1_{transmision} = 90seg$
$t1_{total} = 118seg$

$t2_{autentificacion} = 61seg$
$t2_{transmision} = 58seg$
$t2_{total} = 119seg$

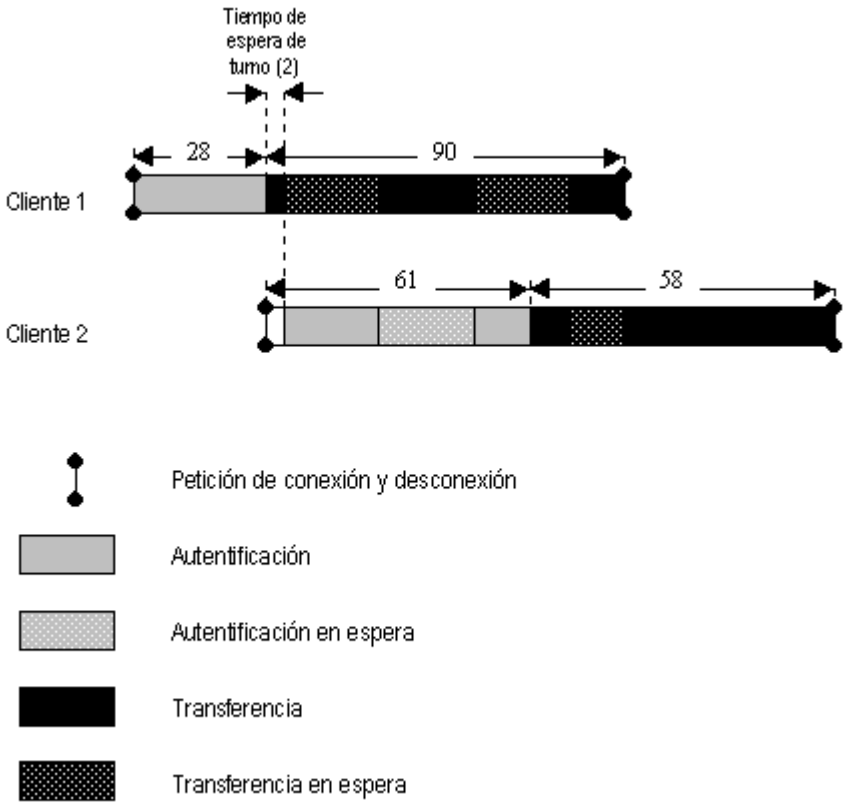


Figura 37: Estudio de DAMA con 2 clientes

$t_{global} = t1_{autentificacion} + t2_{total} = 28 + 120 = 147seg$
$t_{global\_medio} = t_{total}/2 = 74seg$
$t_{sobrecarga} = 147 - 2 * 72 = 3segundos$

$sobrecarga = 100 * (147 - (2 * 72))/(2 * 72) = 2\,\%$

3 Clientes

$t1_{autentificacion} = 28seg$
$t1_{transmision} = 108seg$
$t1_{total} = 136seg$

$t2_{autentificacion} = 62seg$
$t2_{transmision} = 107seg$
$t2_{total} = 169seg$

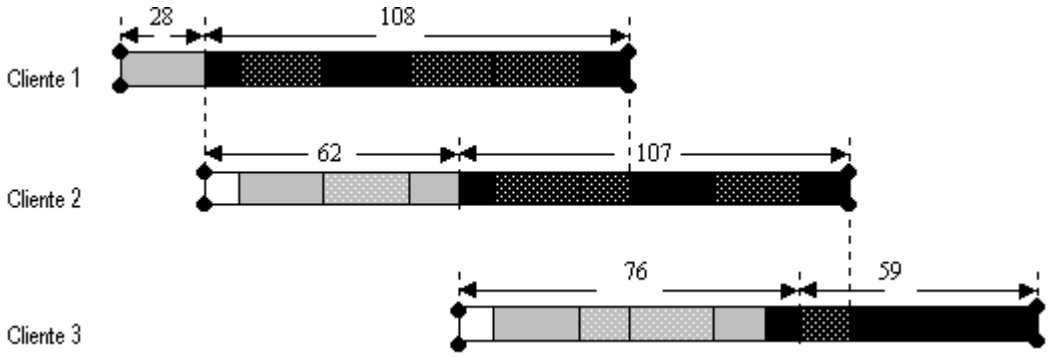


Figura 38: Estudio de DAMA con 3 clientes

$t3_{autentificacion} = 76seg$
$t3_{transmision} = 59seg$
$t3_{total} = 135seg$

$t_{global} = t1_{autentificacion} + t2_{autentificacion} + t3_{total}$
$t_{global} = 28 + 62 + 135 = 225seg$
$t_{global\_medio} = t_{total}/2 = 75seg$
$t_{sobrecarga} = 225 - 3 * 72 = 9segundos$

$sobrecarga = 100 * (25 - (3 * 72))/(3 * 72) = 4\,\%$

11. Conclusión y trabajo futuro

Una vez analizadas las prestaciones que ofrece el nuevo sistema se concluye que los resultados son excelentes. Con este nuevo sistema se logra el máximo rendimiento mediante la consecución de varios objetivos:

- Velocidad del canal radio eficiente gracias al uso del modo AX.25 conectado sin encapsulación IP.
- Compresión y cifrado de los datos mediante la compresión del túnel seguro SSH.
- Implementación del protocolo DAMA, resultando en un sistema con menos colisiones y de menor gasto en energía e infraestructura.

Por todo esto se puede concluir que la capacidad del canal radio está al máximo de la que éste puede ofrecer. En cuanto a la propia estructura del sistema, una mejora que se podría realizar es la integración del sistema en la estructura de protocolos de Windows. De esta forma sería un sistema totalmente transparente con la única diferencia que el dispositivo de acceso al medio es una radio.

Adicionalmente, sin ser objetivos prioritarios del proyecto, se ha dotado a la red de facilidades como la gestión remota por Internet, sistema seguro de autentificación y cifrado de los datos del enlace radio.

En este proyecto no se ha hablado en ningún momento del acceso a Internet (páginas Web) de los clientes. Esto se debe a que en el diseño de necesidades no se creyó conveniente el acceso de los clientes al servicio Web. En cualquier caso, desde el punto de vista técnico, la incorporación del acceso a Internet no presenta ninguna dificultad. Sólo habría que crear una nueva redirección de un puerto en el ordenador cliente hacia un puerto del servidor en el que estuviera instalado un *proxy*, que serviría de pasarela para al acceso a Internet. También habría que hacer que hacer unas modificaciones en el *daemon* cliente para que las hebras de escucha de los puertos de Web no se cerraran hasta pasado un plazo de inactividad. También serían conveniente poner un tiempo de turnos para DAMA menor del que se ha usado para aplicaciones de correo, para impedir ineficiencia por inactividad del canal.

Respecto a las radios hay que decir que los parámetros más importantes que inciden en la eficiencia final son los tiempos de commutación entre transmisión y recepción (*txdelay* y *txtail*), siendo habitualmente mejores (más pequeños) en radios más caras. Para comunicaciones de voz estos valores son prácticamente despreciables pero se vuelve un parámetro a tener en cuenta en comunicaciones digitales.

En el capítulo del análisis de prestaciones de módems se ha visto que se trata de un ámbito en el que no trabajan empresas profesionales, siendo los propios grupos de radioaficionados los que se encargan de su diseño, fabricación y distribución. Esto es, por una parte, positivo, porque normalmente son organizaciones muy abiertas, pero por otra parte repercute en unas prestaciones de los módems no equiparables a productos profesionales.

Finalmente, habría que destacar dos posibles campos sobre los que habría que seguir trabajando para continuar la tarea emprendida. Uno de estos sería la instalación de subredes en la banda de Microondas, que permetirían velocidades de hasta 11 Mbps (con visión directa entre cliente y servidor). Sobre esta opción aún hay que hacer pruebas que confirmen su viabilidad, tanto técnica como económica, en las zonas de actuación.

El otro campo sobre el que se debería actuar es el diseño de un sistema que permita, con una filosofía similar al de este proyecto, la transmisión de información en HF (Alta frecuencia), permitiendo el acceso a localizaciones remotas y aisladas. Hasta el momento este tipo de comunicaciones están limitadas a 300bps, por lo que un proyecto de estas características estaría encaminado en el diseño de nuevos módems que permitieran mayores velocidades de transmisión.

12. Glosario

AX.25 *Amateur X.25*. Versión modificada del protocolo X.25 para comunicaciones vía radio.

CSMA *Carrier Sense Multiple Access*. Acceso múltiple por detección de portadora. Sistema por el cual se observa el canal antes de transmitir para evitar colisiones.

DAMA *Demand Assigned Multiple Access*. Acceso múltiple asignado bajo demanda. Protocolo en que una máquina central asigna turnos a los clientes para evitar colisiones.

DISC *Disconnect*. Paquetes AX.25 de petición de desconexión.

EHAS Enlace Hispano-Americano de Salud. Programa para la introdución de sistemas de comunicación en zonas rurales de paises hispanoamericanos.

GNU *GNU's not Unix*. Proyecto de sistema operativo libre.

IDEA *International Data Encryption Algorithm*. Algoritmo de encriptación basado en claves simétricas.

KISS *Keep It Simple Stupid*. Protocolo simple de comunicación entre un ordenador y la TNC.

POP3 *Post Office Protocol v3* Protocolo para la lectura de mensajes almacenados en un servidor.

PPP *Point-to-Point Protocol*. Protocolo para la conexión entre dos terminales con facilidades como control de errores o compresión.

PTT *Push To Talk*. Señal de entrada de las radio *half-duplex* con la cual se marca si está en modo escucha o en modo transmisión.

REJ *Reject*. Paquete AX.25 que indica petición de retransmisión de paquetes.

RNR *Receiver Not Ready*. Paquete AX.25 para indicar que el receptor no está preparado para recibir más paquetes.

RR *Receiver Ready*. Paquete AX.25 de confirmación de paquetes de información.

RSA Rivest, Shamir, and Adleman. Inventores del algoritmo actualmente más popular de claves asimétricas, basado en la dificultad de factorización de números enteros muy grandes.

SABM *Set Asyncronous Balanced Mode*. Petición de conexión del protocolo AX.25.

SMTP *Simple Mail Transfer Protocol*. Protocolo para la transferencia de mensajes a un servidor.

SSH *Secure Shell*. Protocolo para el establecimiento de conexiones seguras entre ordenadores.

TAPR *Tucson Amateur Packet Radio*. Asociación de radioaficionados.

TDMA *Time Division Multiple Access*. División del canal en *slots* temporales.

TNC *Terminal Node Controller*. Dispositivo *hardware* para la comunicación entre un ordenador y la radio. Normalmente realiza funciones de almacenaje de paquetes y acceso al medio.

UA *Unnumbered Acknowledge*. Paquete AX.25 usado para la aceptación de peticiones de conexión o desconexión.

UUCP *Unix-to-Unix Copy*. Protocolo que permite la transmisión de información entre ordenadores.

X.25 Recomendación CCITT de commutación de paquetes para la interfaz entre equipos DTE y DCE.

Referencias

[1] Página principal del programa AgwPacketEngine. `http://www.raag.org/sv2agw`.

[2] A.Martínez, V.Villarroel, A.Escudero, and F. del Pozo. Enlace hispanoamericano de salud. tecnologías de comunicación para médicos aislados en las zonas rurales de latinoamérica. Technical report, Unión Internacional de Telecomunicaciones, 1999.

[3] Linux AX.25 HOW-TO. `http://www.linuxdoc.org/HOWTO/AX25-HOWTO.html`.

[4] Página principal del grupo de radioaficionados alemán BAYCOM. `http://www.baycom.org`.

[5] William Beech, Douglas Nielsen, and Jack Taylor. AX.25 link access protocol for amateur packet radio v2.2. Technical report, Tucson Amateur Radio Corporation, 1997.

[6] Daniel Pierre Bovet and Marco Cesati. *Understanding the LINUX Kernel: From I/O Ports to Process Management.* O'Reilly, 2nd edition, 2001.

[7] Ramy Card, Eric Dumas, and Frank Menel. *Programación Linux 2.0: API de sistema y funcionamiento del nucleo.* Gestión 2000, 1a edition, 1997.

[8] DAMA - A new method of handling packets. `http://micro2.sch.bme.hu/dama_e.htm`.

[9] Página principal del grupo de radioaficionados DIGIGRUP. `http://www.digigrup.es`.

[10] Página principal del programa EHAS. `http://www.ehas.org/`.

[11] Página principal de Ehtrax25 para windows. `http://www.ampr.torun.pl/packet/ethrax25.html`.

[12] Página principal del programa Flexnet. `www.afthd.tu-darmstadt.de/~flexnet`.

[13] Artículo sobre el diseño de módems compatibles G3RUH. `http://www.amsat.org/amsat/articles/g3ruh/109.txt`.

[14] Versión electrónica de la GPL. `http://www.gnu.org/copyleft/gpl.html`.

[15] Ismael Pellejero Ibañez. Desarrollo de una red de comunicaciones TCP/IP sobre radio. Technical report, Centro Politécnico Superior. Universidad de Zaragoza, 2000.

[16] Página principal de distribución del Kernel de Linux. `www.kernel.org`.

[17] Página principal del programa PartitionMagic. `http://www.powerquest/PartitionMagic`.

[18] Página principal del programa Setup Generator. `http://www.gentee.com`.

[19] Página principal del cliente SSH para Windows SSH32 v1.0. `www.mines.edu/Academic/computer/security/tools/ssh/cigaly.shtml`.

[20] Andrew S. Tanenbaum. *Computer Networks*. Prentice Hall, 3rd edition, 1996.

[21] Página principal de TAPR (Tucson Amateur Packet Radio. `http://www.tapr.org`.

[22] Página principal del módem YAM. `http://www.microlet.com/yam`.

[23] T. Ylonen. The SSH (Secure Shell) remote login protocol. Technical report, Helsinki University of Technology, 1995.

A. Configuración del sistema

A.1. Introducción

Con este nuevo sistema se pretende aprovechar al máximo el canal radio mediante la mejora de eficiencia del protocolo AX.25 y el uso de métodos de compresión. Al igual que en el sistema anterior se permite el uso de aplicaciones de correo estándar en clientes Windows que se conectan a un servidor Linux situado en el Centro de Salud.

Con esta nueva configuración se cambia de forma importante la forma de funcionar que teníamos hasta el momento: en vez de encapsular IP dentro de paquetes AX.25 que emitimos por la radio, enviamos paquetes con datos AX.25 puros, sin encapsulación. Esta forma de operar requiere la instalación de unos programas de escucha (tanto en los clientes como en el servidor) que se encarguen de redirigir todo el tráfico de puertos TCP al canal radio AX.25 y viceversa. A estos programas se les suele llamar *daemons*. Así pues, ahora los clientes no se conectarán a la IP del servidor sino a puertos locales en los que el *daemon* estará escuchando.

En *daemon* instalado en el cliente Windows escucha de forma continua los puertos SMTP y POP3 locales, a los cuales se conectaran los clientes de correo al intentar escibir o recibir mensajes. Una vez el *daemon* detecta esta conexión del cliente en uno de estos puertos, activa un cliente Secure Shell (SSH32) que tenemos previamente configurado para que redirija determinados puertos locales a puertos de la máquina remota. En ese momento nuevamente el *daemon* detecta que el SSH ha intentado abrir una comunicación, la intercepta y pasa a funcionar de pasarela de datos entre el cliente SSH y el verdadero servidor SSH (que se encuentra en el servidor Linux) mandando los datos por la radio con el programa AGWPacketEngine, previamente instalado.

En el ordenador que hace funciones de servidor de correo (con sistema operativo Linux), tenemos activo otro *daemon* que detecta una llamada AX.25, abre una conexión con el puerto del SSH server (puerto 22) y se encarga de pasar los datos entre el socket AX.25 y el socket SSH. El SSH server se encarga de abrir los puertos que corresponda (POP3 o SMTP) en el servidor. En la figura se detalla de forma gráfica toda la estructura.

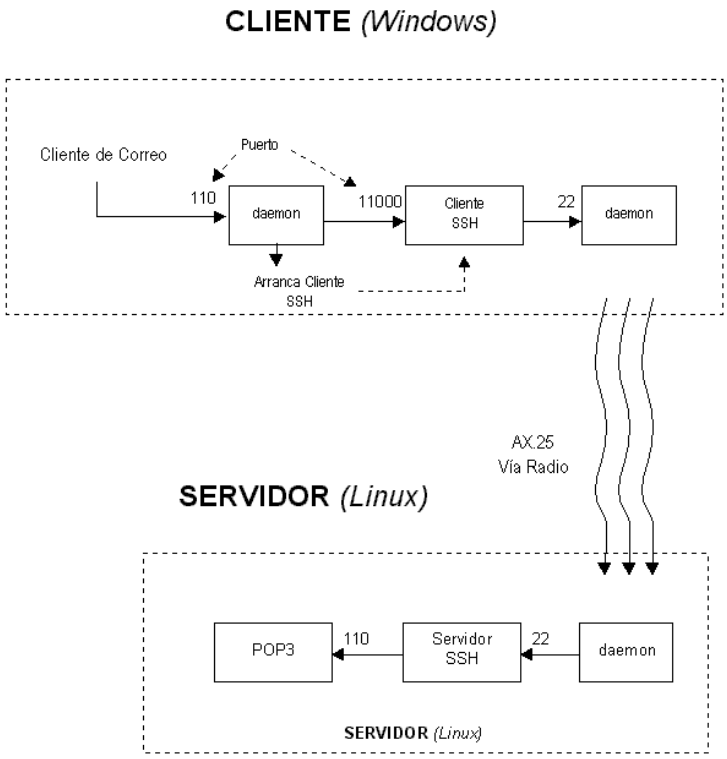


Figura 39: Esquema gereral de los *daemons*

A.2. Configuración del servidor

A.2.1. Kernel de Linux

Lo adecuado es usar un kernel con versión mayor o igual al 2.2.18 pues en esta versión se solucionaron algunos problemas de pérdida de memoria de versiones anteriores. Una vez descomprimido el código fuente del kernel en un subdirectorio (por ejemplo /usr/src/local) se debe configurar para poder usar AX.25 y los dispositivos *hardware* conectados.

(Debian) make menuconfig

(si da error habrá que instalar la biblioteca *ncurses*)

Ahora hay que configurar:

```
Code maturity level options
    [*] Prompt for develpment and/or incomplete code/drivers
Loadable module support
    [*] Kernel module loader
General Setup
    [*] Networking support
Amateur Radio support
    [*] Amateur Radio Support
    [M] Amateur Radio AX.25 Level 2 Protection
    [M] Serial port KISS driver
    [M] PICPAR 9600bps Modem
```

Donde:

```
[*] Forma parte del kernel
[M] Instalado como módulo.
```

Y ya se puede compilar el kernel como de costumbre:

(Debian)

```
make-kpkg clean
make-kpkg --revision ax25_1.0 kernel_image
dpkg -i kernel-image-2.2.18_ax25_1.0_i386.deb
```

Después de rearrancar hay que instalar las aplicaciones y bibliotecas que se usan con AX.25:

(Debian)

```
apt-get install ax25-tools
apt-get install ax25-apps
apt-get install ax25-libs-dev
```

(automáticamente se instalarán también las libax25)

Ahora hay que instalar el paquete con el *daemon* y las utilidades:

(descomprimir en /usr/local) `tar xvfz ax25d.tar.gz`

A.2.2. Ficheros de configuración

Tendremos que modificar los archivos de configuración e incluir la siguiente línea:

/etc/ax25/axports

```
#puerto #CallSign #Vel. #Tamaño #Ventana #Descrip.
tnc     SERTNC   9600  255     7        TNC a 1200bps
```

En el caso de una TNC, en el apartado de velocidad hay que poner la que se usa entre PC y TNC, teniendo que ser, como mínimo, el doble de la velocidad con la que se transmite en el aire que depende de la configuración de la TNC. En este parámetro hay que poner 19200 si queremos transmitir a 9600. Si transmitimos a 1200bps lo habitual es poner 9600.La TNC tiene que estar correctamente configurada para poder comunicarnos con ella a la velocidad deseada mediante unos *jumpers* situados en la parte inferior.

Ahora creamos unos *scripts* de arranque para que se ejecuten de forma automática al arrancar el servidor. Para ello Linux dispone del directorio `/etc/init.d`. Los scripts que se encuentren en este directorio son llamados por unos enlaces simbólicos que se encuentran en los directorios `/etc/rc`**X**`.d` (Donde **X** son los niveles de usuario del 0 al 6). Para la creación automática de los enlaces existe la utilidad `update-rc.d`:

```
update-rc.d nombre_de_script defaults
```

El *script* carga los módulos del protocolo AX.25, los módems y el *daemon* **sshax25d**. Además asigna valores a las constantes que rigen el protocolo AX.25 que se encuentran en `/proc/sys/net/ax25/[disp]` [3]:

- Red antigua con TNCs a 1200bps:
  - Mandamos dos secuencias para asegurar que la TNC esté en modo KISS: `'kiss on'` y `'restart'`.
  - TNC conectada en puerto serie **COM1**.
  - *txdelay* = 350 mseg, *txtail* = 150 mseg. (radios Motorola PRO3100)

- *slottime* = 10 mseg, *ppersist* = 128.
- Tamaño de paquete = 256, Tamaño de ventana = 7.
- *Timeout1* = 15 seg. Este tiempo debe ser mayor al tiempo mínimo de envío de una ventana de 7 paquetes. En caso contrario el PC pediría una retransmisión antes de que la TNC hubiera sido capaz de enviar toda la ventana. El tiempo mínimo es 7x256x8/1200 = 12 segundos.
- *Timeout2* = 5 segundos.
- *Timeout3* = 3 segundos.
- *Idletimeout* = 10 minutos.

ax25tnc

```
# /bin/sh
DAEMON=/usr/local/sbin/sshax25d
NAME=sshax25d
DESC="SSH-AX.25 Daemon"
case "$1"in
  start)
    echo -e -n "\rkiss on\r"> /dev/ttyS0
    echo -e -n "restart\r" > /dev/ttyS0
    insmod ax25
    insmod mkiss
    /usr/sbin/kissattach /dev/ttyS0 tnc 44.0.0.100
    /usr/sbin/kissparms -p tnc -t 350 -s 10 -r 128 -l 150
    echo 256 > /proc/sys/net/ax25/ax0/maximum_packet_length
    echo 7 > /proc/sys/net/ax25/ax0/standard_window_size
    echo 2000 > /proc/sys/net/ax25/ax0/t1_timeout
    echo 500 > /proc/sys/net/ax25/ax0/t2_timeout
    echo 3000 > /proc/sys/net/ax25/ax0/t3_timeout
    echo 60000 > /proc/sys/net/ax25/ax0/idle_timeout
    set -e
    echo -n "Starting $DESC: "
    start-stop-daemon --start --pidfile /var/run/$NAME.pid -b
      --make-pidfile --exec $DAEMON
    echo "$NAME."

  stop)
```

```
        echo -n "Stopping $DESC: "
        start-stop-daemon --stop --quiet --pidfile /var/run/$NAME.pid
          --exec $DAEMON
        echo "$NAME."
        ;;
    esac
    exit 0
```

Hay que hacer notar que la IP que asignamos no se usa en ningún momento: ya no estamos encapsulando IP en AX.25, en el cliente solamente nos estamos comunicando con puertos TCP de la propia máquina (*localhost*). La IP hay que ponerla porque la implementación Linux lo obliga pero no la usaremos.

A.2.3. SSH

El problema que tenemos con AX.25 es que es un protocolo de nivel de enlace por lo que no permite más que una única comunicación entre máquinas, de forma que si queremos comunicarnos con varios puertos TCP habría que hacer una encapsulación (que es lo que hacía Ethrax25).

El problema está en que si usamos el dispositivo AX.25 para transmitir paquetes IP, tenemos otro protocolo (TCP) por encima de AX.25 que de alguna manera hace tareas redundantes a las de AX.25. Un ejemplo claro es el de las retransmisiones: AX.25 ya garantiza que los paquetes se entregan ordenados y sin errores con un sistema de retransmisiones adaptados al medio radio (que normalmente es half duplex, y con tiempos muertos debido al txdelay). TCP tiene su propio sistema de retransmisiones que se superpone al de AX.25 y ralentiza la comunicación considerablemente, haciéndola imposible en la mayoría de casos.

Por ello se ha optado por un esquema alternativo: escribir en el socket AX.25 directamente datos de los puertos TCP locales. En la configuración que teníamos anteriormente (Ethrax25) podíamos usar directamente TCP/IP porque se encapsulaba en paquetes AX.25, pero en la nueva configuración, al usar AX.25 puro, no podemos multiplexar puertos directamente, función requerida si queremos usar los servicios de red habituales: telnet, pop3, smtp, ftp, etc, (en nuestro caso sólo nos interesan correo: SMTP y POP3). El Secure Shell será quien haga la multiplexación, y a la vez nos permitirá usar cifrado y compresión.

Instalamos el paquete SSH:

(Debian) `apt-get install ssh`

Automáticamente se incluirá el demonio *sshd*, que hará funciones de servidor SSH, en un script de arranque.

Con esto ya habremos acabado la configuración del servidor. Sólo quedaría copiar los ficheros de claves RSA que se vayan a usar, en el próximo apartado se explica la forma en que se ha automatizado este proceso.

El protocolo RSA nos da un sistema seguro, rápido y cómodo para la autenticación con el cliente. Se basa en la generación de un par de claves, una pública y otra privada. La pública la ponemos a disposición de todo el mundo (tendremos una copia en el servidor) mientras que la privada tiene que estar guardada en el ordenador sin que ningún otro usuario pueda acceder a ella. La clave pública será la que conozca el servidor y la privada nos permitirá autentificarnos ante él.

A.3. Configuración del cliente

La configuración en los clientes Windows es algo más sencilla. Las pruebas se han hecho sobre Windows 98 (segunda edición). Es muy probable que en Windows 95 no funcione (por problemas con librerías antiguas) y en Windows 2000 debería funcionar, aunque no se ha probado. Lo primero que hay que hacer es ir a:

Panel de control→Red

y quitar todas las configuraciones antiguas referentes a TCP/IP. Después hay que instalar el acceso telefónico a redes:

Agregar→Adaptador→Microsoft→Adaptador acceso telef. redes.

Necesitamos instalar este dispositivo de acceso virtual porque Windows necesita un adaptador sobre el que instalar el protocolo TCP/IP, aunque luego no haremos uso de él. Ahora ya podemos añadir el protocolo TCP/IP:

Agregar→Protocolo→Microsoft→TCP/IP

Y reiniciamos. No hace falta configurar nada, ni DNS, ni IP.

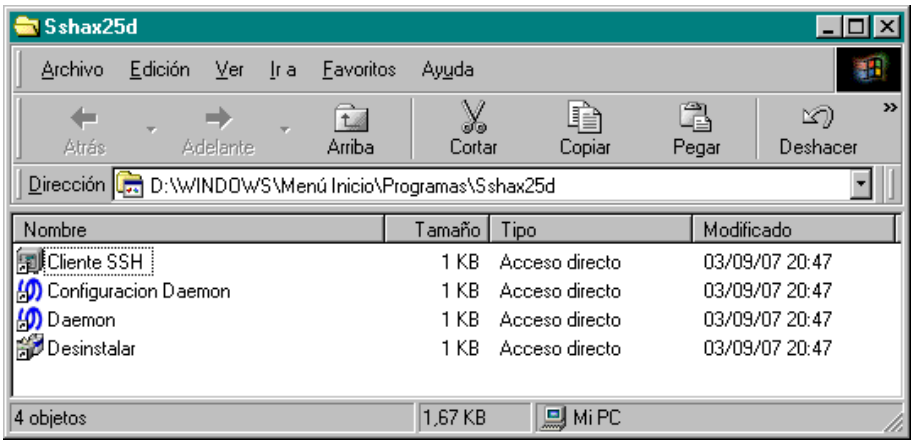


Figura 40: Carpeta del *daemon* de Windows

Instalamos el paquete en el que se incluye el *daemon*, el SSH y el AGWPacketEngine (programa que implementa el protocolo AX.25, sustituto de Ethrax25) en el directorio que se quiera.

Una vez ejectuado y finalizado el programa de instalación (`wintnc12.exe`) se ha creado automáticamente una carpeta con los accesos directos al *daemon* y se abre una ventana de configuración que permite la modicicación de los siguentes parámetros:

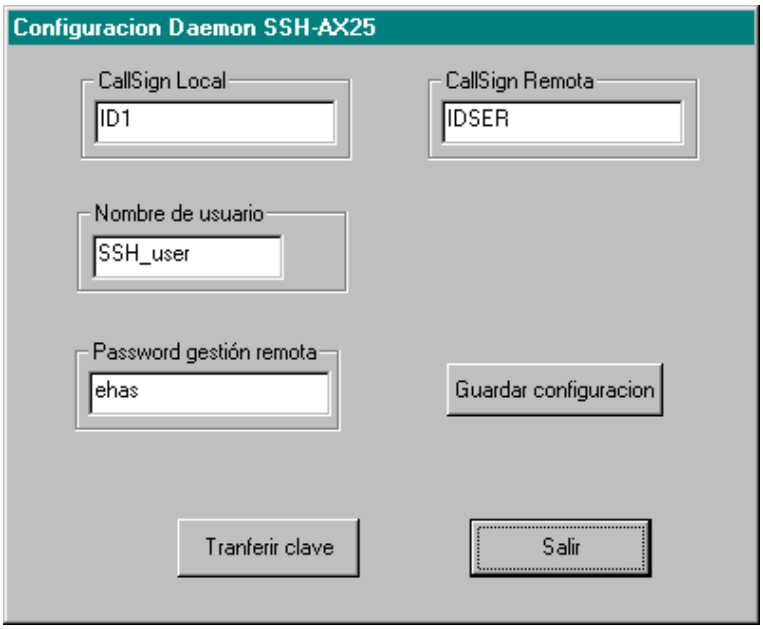


Figura 41: Opciones del *daemon* de Windows

- **CallSign Local**: Indicativo del cliente
- **CallSign Remota**: Indicativo del servidor
- **Password de gestión remota**: Contraseña para entrar en el cliente Windows desde un servidor Linux mediante la utilidad **wtelnet**
- **Nombre usuario SSH**: Nombre del usuario en cuya cuenta se autentificará el cliente SSH32.
- **Guardar configuración**: Se guarda la configuración modificando en el fichero `daemon.cfg` los callsigns y la contraseña, y modificando el registro donde se almacena la configuración del SSH32. Si la clave para ese usuario SSH no existía se crea. La creación de las claves se hace con el programa `SSH-KEYGEN.EXE`, de distribución gratuita.
- **Transmitir clave**: Se transfiere la clave pública (`[nombre_de_usuario.PUB]` al servidor. La transferencia se hace gracias al uso del programa SCP (*Secure Copy*). Para que la conexión sea posible se activa el *daemon* mientras dure la transferencia.

Una vez finalizado este proceso de instalación y configuración el *daemon* queda instalado en el registro de arranque de Windows, por lo que arranca de forma automática cada vez que se enciende el ordenador. Para hacerlo se ha añadido automáticamente el path de *daemon* en la clave de registro siguiente:

```
HKEY_LOCAL_MACHINE\Software\Microsoft\CurrentVersion\Run
```

También se incluye un icono para desinstalar la aplicación completa, excepto los ficheros que se hayan añadido posteriormente (las claves públicas y privadas), cuyo borrado es opcional.

A.3.1. Configuración de Netscape

En el Messenger hay que poner la siguiente configuración:

Servidor POP3 entrada: *localhost*

Usuario POP3: El usuario de correo en el servidor

Servidor POP3 salida: *localhost*

Es conveniente deshabilitar la opción *Comprobar mails cada X minutos* para evitar accesos a radio no pedidos.

Si se ha realizado todo el proceso con éxito, las diferencias serán que al rearrancar habrá aparecido un nuevo icono en el *system tray* (se trata del AGW) y habrá un proceso residente (el *daemon*) que se puede ver si está activo haciendo Ctrl-Alt-Del, debiendo aparecer en la lista de procesos.

Ahora ya podemos hacer **Obtener Mensaje** y **Enviar Mensaje** como siempre. El *daemon* permite leer y mandar un mensaje a la vez, pero no leer o escribir dos o más correos a la vez. En el momento de dar a **Obtener Mensaje** o **Enviar Mensaje** aparecerá en la barra de tareas el SSH durante la fase de autentifiación . En el momento en que desaparezca de la barra de tareoas y aparezca en el *system tray* querrá decir que el proceso de autenticación ha finalizado y se estáa estableciendo comunicación con los puertos SMTP ó POP3. Cuando la comunicación finalice, el icono de SSH también desaparecerá, se cerrará el túnel SSH y con él la conexión AX.25.

A.4. Gestión remota

Los usuarios finales del sistema son personal médico con conocimientos informáticos limitados, por ello se creyó interesante dotar a la red de un sistema de gestión remota posible desde cualquier ordenador conectado a Internet.

El objetivo es no sólo permitir la administración del servidor DAMA o del servidor del centro de salud, lo cual es sencillo por tratarse de máquinas Linux, sino también de los clientes Windows. Esto último plantea la dificultad de que Windows no incorpora ningún método de gestión remota, ni siquiera el acceso a un intérprete de órdenes para realizar operaciones básicas.

De tal forma, se pensó que el mismo *daemon* que se encargaba de interceptar la conexión a los puertos de SMTP y POP3, podía servir para recibir peticiones de conexión AX.25 del servidor e iniciar la ejecución de un intérprete de órdenes, redirigiendo la salida y entrada estándar al canal radio.

Otra dificultad que habrá que solventar es que no conocemos la dirección IP del servidor del Centro de Salud que se conecta a Internet por lo que habrá que diseñar un modo de saberla.

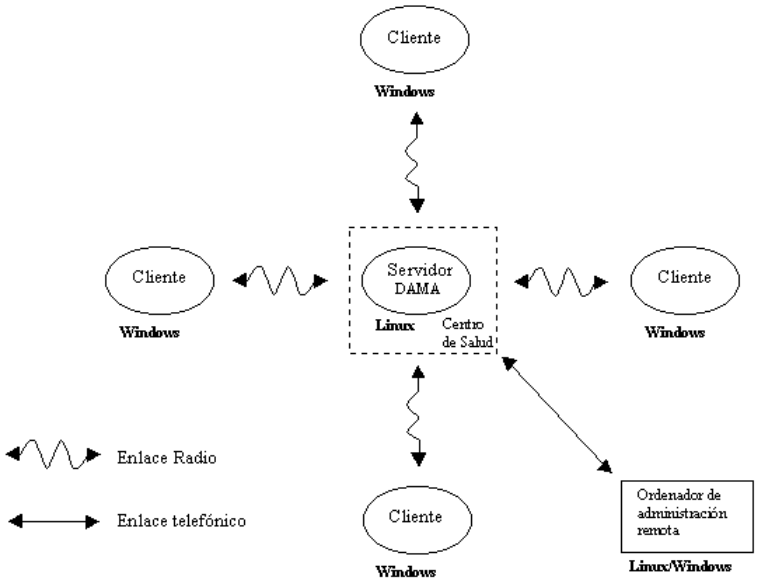


Figura 42: Arquitectura de red. Servidor DAMA en el Centro de Salud

A.4.1. Esquema del sistema de gestión remota

Como ya vimos en la Introducción, podemos tener dos escenarios dependiendo si el servidor DAMA es el propio ordenador del Centro de Salud o no:

- Servidor DAMA en el centro de Salud. Figura 42
- Servidor DAMA dedicado. Figura 43

A.4.2. Gestión del cliente Windows

Es importante destacar que el cliente no podrá ni escribir ni leer correo durante el proceso de gestión remota. Durante este proceso el *daemon* no está escuchando el puerto SSH así que cualquier intento de lectura o escritura de mail dará un error en el cliente de correo.

Con este sistema se permite realizar las operaciones que se pueden hacer normalmente en una *shell* de Windows. Se consideró que en algunos casos podía ser interesante realizar una transferencia de ficheros entre cliente y servidor, y se estableció un sistema simple y sencillo para ello. Si la orden que se entra en el

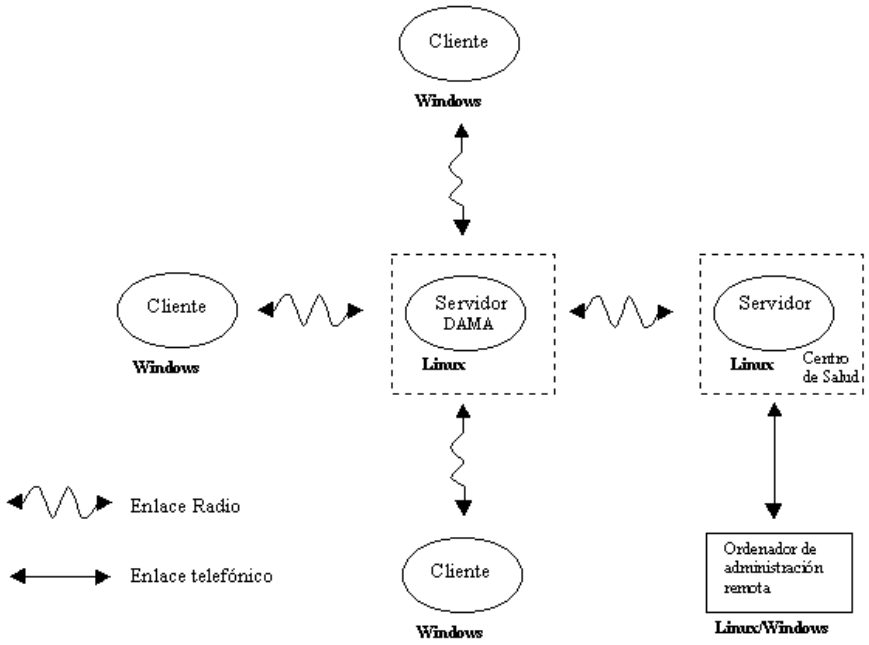


Figura 43: Arquitectura de red con servidor DAMA dedicado

prompt empieza por algún símbolo del tipo ">"ó "<", se intercepta y se pasa a modo de transferencia de fichero. Veamos la sintaxis en detalle:

>**nombre_fichero.** Envía un fichero desde el directorio actual del servidor Linux al directorio donde está instalado el *daemon* en Windows.

<**nombre_fichero.** Envía un fichero desde el directorio de instalación del *daemon* de Windows al directorio actual del servidor Linux.

Estos caracteres son introducidos en el programa **wtelnet** que realiza la conexión desde el servidor. Dicho programa se analiza con profundidad más adelante en este mismo capítulo.

Servidor DAMA en Centro de Salud

Este es el caso más simple. Hasta el Centro de Salud se puede entrar con un cliente `ssh` de la forma habitual. Para poder entrar después en el cliente Windows, se ha creado el programa `wtelnet`:

Uso: `wtelnet puerto callsign_origen callsign_destino`

Se introduce la contraseña y se abrirá un *shell* para la administración de cliente.

Servidor DAMA dedicado

En esta configuración es necesario un salto adicional (ver figura 43) que en el caso anterior. Veamos la secuencia:

1. Desde el ordenador en Internet entramos en el servidor del Centro de Salud mediante un conexión SSH.

2. Desde el servidor del Centro de Salud hay que entrar en el servidor DAMA. Para ello se ejectua el programa residente `sshax25cliente`. Este programa intercepta las conexiones de los clientes SSH en el puerto 22000. No se puede usar el puerto 22 porque este ya está siendo usado como servidor SSH. Para activar el `sshax25cliente` hay que seguir la siguiente sintaxis:

   ```
   sshax25cliente puerto indicativo_serv indicativo_serv_dama
   ```

   Este *daemon* puede quedar siempre activado porque no interfiere el funcionamiento normal del servidor del Centro de Salud. Así pues, se puede añadir su ejecución al *script* de arranque correspondiente a dicho servidor.

   Una vez el *daemon* está en memoria podemos realizar una conexión a él:

   ```
   ssh usuario@localhost -p 22000
   ```

   Vemos que hay que decirle explícitamente que el puerto a conectarse es el 22000, en caso contrario lo haría en el puerto 22 donde no haría más que entrar en la propia máquina.

   El *daemon* se encarga de dirigir la petición del cliente SSH al canal radio. Nos identificamos ante el servidor DAMA y una vez ya hemos entrado en él, nos encontramos en la misma situación que el apartado anterior: sólo queda usar `wtelnet` para entrar en el cliente Windows.

Si nos fijamos de nuevo en la figura 43 veremos que estamos usando simultáneamente dos enlaces radio:

- Servidor Centro de Salud $\longleftrightarrow$ Servidor DAMA
- Servidor DAMA $\longleftrightarrow$ Cliente Windows.

Se hace imprescindible, pues, deshabilitar el protocolo DAMA. Si no lo hicieramos, el servidor DAMA otorgaría turnos de forma secuencial a estos dos enlaces y cualquier petición tardaría un mínimo del tiempo de un turno en tener efecto, lo que haría el proceso de gestión realmente tedioso.

Cabe resaltar que el hecho de desactivar el protocolo DAMA en este caso concreto (administración remota) no afecta de forma negativa a la comuncación ni provoca colisiones de paquetes. Si se analiza el tráfico en una administración se ve que un proceso habitual de **petición-respuesta** no provoca transmisiones simultáneas de clientes a servidor DAMA, que es el caso en que las colisiones serían inevitables.

A.4.3. Gestión del servidor Linux desde un cliente Windows

Hasta ahora sólo nos hemos centrado en la gestión que desde el servidor Linux se puede hacer en los clientes Windows. También se puede hacer una gestión remota del servidor Linux desde los clientes Windows, mediante el uso directo del programa SSH32. Para ello se ejecuta directamente esta aplicación (sin activar ningún cliente de correo) y el *daemon* se encargará de redirigir la conexión por el enlace radio y se abrirá una *shell* en el servidor.

Si nos interesara hacer transferencias entre el cliente y el servidor también podemos usar el programa `SCP` (secure copy) incluído en el paquete del *daemon* de Windows. El `scp` tiene la siguiente sintaxis:

```
scp.exe origen destino
```

El parámetro que sea un fichero remoto tendrá la siguiente sintaxis:

```
usuario@host:/[ruta]/fichero
```

Así pues, si quisieramos copiar el fichero `config.sys` al directorio `home` del usuario `prueba` haríamos:

```
scp c:\config.sys prueba@localhost:/home/prueba/config.sys
```

Notar que como siempre el *host* a poner en la orden es *localhost*, de esta forma el **scp** conecta con el puerto 22 y el *daemon* de Windows empieza su función de pasarela. En estos casos de administración los *hebras* de SMTP y POP3 no intervienen para nada.

A.5. Alcance de la gestión remota

Vamos a hacer un repaso de las posibilidades que nos permite el sistema de gestión remota descrito en el presente apartado:

- Copiado, borrado, renombrado de ficheros y en general cualquier operación realizable desde la línea de órdenes MsDos.
- Transferencia de ficheros entre el servidor y el cliente (mediante los caracteres > y <)
- Edición de ficheros de texto. No es posible el uso de programa de pantalla completa para la edición de ficheros. Por ello en el paquete se incluye un antiguo programa de edición de línea presente hasta la versión 3.0 de MsDos (ver Anexo H)
- Edición del registro. El programa REGEDIT permite la exportación e importación de claves. Las opciones disponibles son las siguientes:

  regedit /e registro.txt Exporta el registro entero a un fichero de texto.

  regedit /e registro.txt <clave de registro> Exporta una clave o un árbol de claves del registro a un fichero de texo . Permite una fácil edición remota (con EDLIN) pues el fichero generado es pequeño. Ej:

  regedit registro.txt. Actualiza el registro de Windows con los cambios que haya en el fichero `registro.txt`, tanto si éste contiene el registro entero o sólo una parte.

  Para más detalles consultar el Anexo I.

- Restauración automática de imágenes mediante el programa PQDI de PowerQuest (ver próximo apartado y Anexo G)

A.5.1. Restauración automática de imágenes

Como la gestión remota tiene sus limitaciones y se puede dar la posibilidad de una corrupción general del sistema Windows o sus aplicaciones, se ha implementado un sistema que permite la restauración del sistema operativo y de las aplicaciones instaladas de forma remota.

Estructura de particiones de los clientes Windows

Para poder hacer una restauración selectiva de particiones se ha establecido la siguiente partición del disco duro:

Partición 1. Unidad C: Sistema Operativo y Aplicaciones Windows.

Partición 2. Unidad E: Documentos de usuario.

Partición 3. Unidad F: Imágenes para restaurar.

De esta forma podremos restaurar solamente las particiones que interesen sin destruir datos de usuario.

Scripts de restauración automática

El programa PQDI de PowerQuest incluido en la aplicación Partition Magic [17] incorpora un control gráfico mediante ventanas para la creación y restauración de imágenes. La gestión remota no permitiría la ejecución e interacción con un sistema de estas características, pero PQDI también incluye un sistema automático mediante el uso de *scripts*. En este punto se presenta el problema que, como es obvio, la restauración de las particiones sólo se puede realizar en modo MsDos puro. Dado que el *daemon* que permite la gestión remota sólo está activo en Windows, hay que implementar algún método para ello.

El método escogido ha sido la sustitución del fichero de arranque (`autoexec.bat`). Cuando el ordenador rearranque y ejecute el nuevo `autoexec.bat`, se ejecutarán las órdenes necesarias para la restauración del sistema completo.

El *script* de restauración sería el siguiente:

```
SELECT DRIVE 1
DELETE PARTITION 1
SELECT FREESPACE FIRST
SELECT IMAGE ALL
RESTORE
REBOOT
```

y en el nuevo `autoexec.bat` incluiremos esta línea:

```
G:\PQDI /CMD=G:\SCRIPT1.TXT /IMG=G:\IMAGEN.PQI
```

Así pues, la secuencia completa para la restauración de imágenes sería la siguiente:

1. Entrar en el cliente Windows de la forma descrita anteriormente.

2. Incluir la nueva línea en el `autoexec.bat`. Para ello se puede usar el programa EDLIN o substituir el `autoexec.bat` por uno alternativo que incluya la mencionada línea.

3. Rearrancar el cliente. Para ello se hace uso de la biblioteca `SHELL32.DLL` de Windows. Dentro de esta biblioteca encontramos la función *SHExitWindowsEX* con los siguientes parámetro:

   1: LOGOFF (Sale de la sesión actual)
   2: REBOOT (Reinicia el ordenador)
   4: FORCE (Fuerza la finalización de los programas activos)
   8: SHUTDOWN (Apaga el ordenador)

   Los parámetros pueden superponerse: así, nuestro programa necesitaría:

   1: LOGOFF
   2: REBOOT
   4: FORCE

   Párametro $= 1 + 2 + 4 = 5$

   Así pues, la orden a ejecutar sería:

   ```
   RUNDLL32.EXE shell32.dll,SHExitWindowsEx 5
   ```

4. En este momento el ordenador se reiniciará y perderemos la conexión establecida con él. Después del tiempo estimado de restauración se podrá volver a conectar con el cliente y se podrá comprobar si la estructura del disco duro es la correcta.

A.6. Determinación IP del Centro de Salud

Hasta ahora no nos hemos preocupado de un aspecto importante: desconocemos la dirección IP del primer ordenador en el que hay que entrar en el proceso de gestión remota, que es el servidor Linux del Centro de Salud (que puede ser a su vez servidor DAMA). Este ordenador, que no dispone de conexión permanente a Internet, se conecta de forma periódica para llevar y traer el correo hacia el exterior.

De lo que si disponemos, como es lógico, es del número de teléfono del Centro de Salud al que está conectado el ordenador a través de un módem. Con una llamada a través de otro módem, podriamos entrar en el servidor, pero entonces se trataría de una llamada nacional o internacional y tendría un coste muy alto. Por ello se ha diseñado un sistema con el que indicamos al servidor que se debe conectar a Internet (con lo que ya se trataría de llamada local) y que debe informar de su dirección IP (que será asignada dinámicamente en cada conexión por el protocolo PPP). En primer lugar, habrá que llamar mediante un módem al Centro de Salud:

```
cu -l /dev/ttyS2
atdt #telefono
login: rellamada
password: ******
```

Una vez esto, la dirección IP del ordenador que realizó la petición (que será el ordenador del Centro de Salud) quedará almacenada en el fichero `/tmp/rellamada` de una máquina permanentemente conectada a Internet (central.ehas.org) donde la podremos consultar. Sabiendo la IP ha podremos conectarnos de la forma habitual mediante un cliente SSH.

B. Descripción del protocolo AX.25

C. Configuración AX.25 en Linux

D. Optimización del protocolo AX.25

E. Tutorial de SSH

E.1. What Is SSH?

The traditional network services like ftp, pop or telnet are convenient but inherently unsafe, since they all make you send a password and data in clear text over an increasingly unsafe network. It is almost a no-brainer to intercept these services and copy transferred data. Furthermore the authentication of the server is weak: the services are open to so-called *man-in-the-middle* attacks, where an intruder pretends to be the server and thus receives all data the client is sending.

Enter SSH (Secure SHell). By using SSH, you encrypt the traffic and you can make *man-in-the-middle* attacks almost impossible. It also protects you from DNS and IP spoofing. As a bonus, it offers the possibility to compress the traffic and thus make transfers faster. SSH is a very versatile tool: not only does it replace telnet, you can also *tunnel* services like ftp, pop and even ppp via it.

SSH implementations exist for almost all major operating systems.

The original SSH has been developed by a Finnish company. Due to copyright restraints and patented algorithms, the Free Software world now uses OpenSSH, a free SSH workalike.

SSH consists of a client-server pair like all the other services. Every system administrator worth his money runs an SSH server. If your remote host doesn't run SSH, you should really think about switching to a host which does. A site which doesn't run an SSH server shows a serious lack of interest in network security.

SSH comes in two major, partly incompatible versions, 1.x and 2.x. You won't be able to connect to an SSH 1.x server with an SSH 2.x client. OpenSSH 2.x supports both versions.

E.2. How SSH Authentication Works

Seen from the client level, SSH provides two levels of authentication.

The first level allows you to connect from any machine to a SSH server, as long as you know the password of the account on the remote machine. This encrypts any traffic sent via SSH, but doesn't provide a strong mechanism to authenticate the host you are connecting to. Another host could intercept your connection by

pretending to be the host you want to connect to (*man-in-the-middle-attack*).

The second level relies on the key mechanism: you create your own keypair and put the public key onto the server. Now if you connect to the SSH server, your client sends a request to the server for authentication using your keys. The server looks up the public key in your remote home directory, and compares both keys. Then it sends an encrypted challenge to the client. This challenge is decrypted on the local machine using the private key and sent back to the server. Using this method, you will have to know the password of your key (if you choose to use one). In contrast to level one, this password will not be sent over the network. Level two authentication doesn't use any passwords at all. This scheme not only encrypts any travel sent via SSH, but also makes *man-in-the-middle* attacks next to impossible. This login process usually takes ten seconds.

E.3. Installing And Testing OpenSSH

Due to U.S. restrictions on exporting strong cryptography, the OpenSSH packages are not included in LM. You can download them from one of the servers listed on LM's crypto apps page.

You will need these packages:

openssl

openssh

openssh-clients

and, if you want to run an ssh-server, openssh-server

Of course the truly paranoid will prefer building them from source, but usually RPMs will do ;). Note that 7.1 OpenSSH RPMs will not install on 7.0.

To test your installation, connect to a SSH server:

ssh -l [your accountname on the remote host] [address of the remote host]

If this works, you will receive a message like this:

The authenticity of host [hostname] can't be established.

Key fingerprint is 1024 5f:a0:0b:65:d3:82:df:ab:44:62:6d:98:9c:fe:e9:52.

Are you sure you want to continue connecting (yes/no)?

SSH tells you that it doesn't know this host, which is nothing to worry about, since you are connecting for the first time. Type yes. This will add the *fingerprint* of this host to */.ssh/known_hosts*. Future connects to the host will not display this message. Then SSH will prompt you for your account password on the remote machine. Type it, press ENTER et voila, you've established your first SSH connection! Now proceed just like you would in a telnet session.

Installing an OpenSSH server is easy, too. Just install the RPM. During the installation, you will get a message like this:

Generating RSA keys: .ooooooO.....ooooooO

Key generation complete.

Your identification has been saved in /etc/ssh/ssh_host_key.

Your public key has been saved in /etc/ssh/ssh_host_key.pub.

The key fingerprint is:

5f:a0:0b:65:d3:82:df:ab:33:52:6f:89:9a:fe:e9:52 root@[local machine]

Generating DSA parameter and key.

Your identification has been saved in /etc/ssh/ssh_host_dsa_key.

Your public key has been saved in /etc/ssh/ssh_host_dsa_key.pub. The key fingerprint is:

64:85:4a:da:cf:74:94:0d:5a:93:cf:f2:62:ed:07:26 root@[local machine]

This message indicates that the key, which identifies your machine to clients, has been generated. It is a very good idea to keep external backup copies of these keys!

Start the server with

service sshd start

That's all. Now external ssh-clients can connect to the machine (provided they have an account on your machine, naturally). If you want the SSH server to be started every time you boot the machine, run

chkconfig –add sshd

E.3.1. Generating Your Own Keypair

Generating and distributing your own keys has two advantages: you protect yourself from *man-in-the-middle* attacks (e.g. by a machine which fakes the fingerprint of the remote host) and you can use one password for all the servers you connect to.

The key is generated by the command

ssh-keygen

If all the remote servers you want to connect via key authentication to use SSH 2.x, you can change that to

ssh-keygen -d

However, SSH2 servers will also accept SSH1 keys, so you might be better off by using the first command.

ssh-keygen will invoke the following dialogue:

Generating RSA keys: .....oooooo0.....oooooo0

Key generation complete.

Enter file in which to save the key (/home/[user]/.ssh/identity):

[Just hit ENTER here unless you already have another key with that name, e.g.

for a different SSH version]

Created directory */home/[user]/.ssh*.

Enter passphrase (empty for no passphrase):

[The entered passphrase will not appear on the screen.]

Enter same passphrase again:

[The passphrase is not recoverable. If you forget it, you will have to generate a

new keypair.]

Your identification has been saved in /home/[user]/.ssh/identity.

[This is your private key.]

> Your public key has been saved in /home/[user]/.ssh/identity.pub.
>
> The key fingerprint is: 2a:dc:71:2f:27:84:a2:e4:a1:1e:a9:63:e2:fa:a5:89 [user]@[local machine]

ssh-keygen -d basically does the same, but saves the keypair by default to */home/[user]/.ssh/id_dsa* (private key) and */home/[user]/.ssh/id_dsa.pub* (public key).

Now you have a keypair: a public key to distribute to all the remote machines you want to ssh to and a private key, which is the heart of the authentication process. Which means: noone should ever be able to access your private key! ls -l /.ssh/identity or ls -l /.ssh/id_dsa should always show these permissions:

> -rw———-

If you suspect your private key has been compromised, do not hesitate to generate a new pair. You will then have to distribute your new public key again, of course.

E.3.2. Distributing Your Key

On each server you need an SSH connection to, create a .ssh subdirectory in your home directory. Into this directory, copy the local file */.ssh/identity.pub* and rename it to *authorized_keys* (or - for SSH2 keys - to *authorized_keys2*). Now execute

> chmod 644 .ssh/authorized_keys (or chmod 644 .ssh/authorized_keys2 for SSH2 keys)

Do not forget this step, SSH won't work if the *authorized_keys* file is writable by anyone other than you!

authorized_keys can hold more than one public key, in case you want to connect to the remote server from a different machine. In this case you have to generate a new keypair on the machine, copy the content of the local *identity.pub* file and paste it into the remote *authorized_keys* file. Of course you should only do that if you have an account of your own on the client machine and the key is password protected! Furthermore, don't forget to remove the keypair when you no longer need it. Simply put: it's better not to use key-based authentication on untrustworthy machines ;).

E.3.3. Keeping Keys In System Memory

This method comes in handy when you usually connect to more than one machine during a session. The trick is to run applications which are automatically authenticated. This is achieved by a combination of the programs ssh-add and ssh-agent. man ssh-add reads:

"The authentication agent must be running and must be an ancestor of the current process for ssh-add to work."

ssh-agent executes several commands, creates an PID file and sets some system environment variables. To make this work, you need the shell's eval command:

eval $(ssh-agent)

You should now see a message like

Agent pid [number]

Enter

ssh-add

in the same terminal and the key will be loaded into memory, asking you for the password, if you have protected the key with one. Now you can start SSH sessions from this terminal without having to give any passwords, provided you set up the SSH *config* file accordingly, which will be discussed on the next page.

If you want to be *ssh-agent* the *ancestor* of all virtual terminals in a session, add the command eval (ssh-agent) to your personal X startup files, either /.xinitrc, if you are starting X from the console, or */.xsession*, if you are booting directly into X. Then run ssh-add and all terminals you will open during this session will be authenticated.

Note that all this isn't necessary for LM 7.1 and later, if you boot directly into X: */etc/X11/Xsession* automatically starts the agent if there is an SSH key in the user's */.ssh* directory. (Thanks, Denis ;-)).

Other useful options are

ssh-add -l which lists the key(s) currently kept in memory, and ssh-add -d which removes an identity from the system memory.

E.4. Configuring The Client

OpenSSH knows three configuration levels: command line options, user configuration file, and system-wide configuration file (*/etc/ssh/ssh_config*). Options given on the command line prevail over configuration file options, options given in the user's configuration file prevail over those in the system-wide configuration file. All commandline options are available as configuration file options. Since there is no user configuration file installed by default, copy and rename */etc/ssh/ssh_config* to */.ssh/config* (or edit */etc/ssh/ssh_config* in place as *root*).

The standard configuration file looks like this:

> [lots of explanations and possible options listed]
>
> Host * ForwardAgent no ForwardX11 no FallBackToRsh no

[Available options are explained in man ssh, chapter CONFIGURATION FILES]

The configuration file is read sequentially, i.e. the first setting that matches a pattern *wins*. Let's say you have an account at www.foobar.com and your account name is *bilbo*. Furthermore you want to use the *ssh-agent* - *ssh-add* combo (discussed on the previous page) as well as data compression to speed up transfers. And since you are too lazy to type the full hostname every time, you want to use 'fbc' as an abbreviation for 'www.foobar.com'. Your configuration file should then look like this:

> Host *fbc HostName www.foobar.com User bilbo ForwardAgent yes Compression yes
>
> Host * ForwardAgent no ForwardX11 no FallBackToRsh no

Next time you enter ssh fbc, SSH will look up the full hostname, use your user name to login and authenticate using the key managed by the *ssh-agent*. It can't get much easier than that, can it? ;)

SSH connections to all other hosts will still use the *paranoid* default settings, the configured accounts only those paranoid settings which haven't been explicitly turned off in their configuration or on the command line. In the example above, an SSH connection to www.foobar.com will have these options set to *yes*: *ForwardAgent* and *Compression*, these options however will still be set to *no* unless overridden by command line arguments: *ForwardX11* and *FallBackToRsh*.

Further options you might want to have a look at include:

CheckHostIP yes This option performs an additional IP address check on the remote host to prevent DNS spoofing. CompressionLevel The compression level ranges from *1* (fast) to *9* (best). Default is '6'. ForwardX11 yes You will need this option to run remote X applications locally. LogLevel DEBUG This option comes in handy when you've got trouble with your SSH connection. The default setting is INFO.

E.5. Configuring The Server

SSH server configuration is done via the file */etc/ssh/sshd_config*, options are explained in the file itself and in man sshd. Note that OpenSSH does not have different configuration files for SSH 1.x and 2.x.

Among the default options you might want to have a look at, are:

PermitRootLogin yes. A preferable option might be PermitRootLogin without-password, which disables *root* logins from machines without a matching keypair. Setting this option to *no* disables *root* logins completely and you have to use su from a user account.

X11Forwarding no. Change this option to *yes* to allow your users to run X apps on your machine. Furthermore, disabling this option doesn't improve your server's security since üsers can always install their own forwarders"(man sshd).

PasswordAuthentication yes. Setting this option to *no* will only allow SSH logins using the key mechanism. This might annoy users who are logging in from different machines frequently but is a boost to server security (password-based authentication schemes are weak).

E.6. Copying Files With scp

SSH gives you access to a set of commands and a shell on a remote machine. By itself, it does not enable you to copy files, it however provides the scp command. Say you want to copy a file called *dumb* from the current directory of the local machine to your home directory on a remote machine called www.foobar.com. Your account name on the remote machine is 'bilbo'. The command would be:

scp dumb bilbo@www.foobar.com:.

To copy it back:

scp bilbo@www.foobar.com:dumb .

scp calls SSH to do the login, then it copies the file and then calls SSH again to close the connection.

If your */.ssh/config* already contains a configuration for your account on www.foobar.com like

Host *fbc HostName www.foobar.com User bilbo ForwardAgent yes

then you can replace 'bilbo@www.foobar.com' with 'fbc': scp dumb fbc:.

scp assumes your home directory on the remote machine to be your current working directory, so if you are using relative paths for the remote machine, they have to be relative to the location of your home directory on that machine. Using the -r switch for *scp*, you can also copy directories recursively. *scp* also allows you to copy files between remote machines.

Now you might be tempted to try something like this: you open an SSH connection to www.foobar.com. Once you are logged in, you type scp [local machine]:dumb . to copy the local file *dumb* to the remote server you are currently logged in. Most likely you will get a message like

ssh: secure connection to [local machine] refused

What has happened is that you executed the remote version of *scp*, and it tried to connect to an SSH server running on your machine ... So remember to run *scp* always from a local terminal, unless your machine also runs an SHH server.

E.7. Tunneling Basics

Tunneling in SSH works by *port forwarding*: you establish a connection between a (non-privileged) local port and the port which runs the service to be tunneled on the remote machine (have a look at */etc/services* for a listing of standard ports). Then you connect to the local port. All requests aimed at the local port are forwarded to the remote port via SSH, and thus encrypted. Tunneling only works if the remote host runs an SSH server, naturally. To check if a remote server runs an SSH server, *telnet* to port 22 of the remote host:

telnet [full name of remote host] 22

You should receive a message stating the version of the remote SSH server running. If you get a message like

telnet: Unable to connect to remote host: Connection refused

then the remote host does not run an SSH server.

The port forwarding syntax follows this scheme:

ssh -f [username@remote host] -L [local port]:[full name of remote host]:[remote port] [some command]

You can forward multiple ports and you can configure frequently used forwardings in */.ssh/config* using the *LocalForward* option. You can also forward remote ports to local ports. To forward privileged ports, you need to be *root*.

E.7.1. Tunneling POP

You can use the Post Office Protocol to get your mail from your mail service provider (e.g. your ISP, university or employer). Tunneling it through SSH should mainly prevent network sniffers from detecting your POP password. As a bonus, you can use SSH's compression mechanism to make mail transfers faster.

Say you have got an POP account at pop.foobar.com, your username is 'bilbo' and your POP-password is 'topsecret'. The command to establish an SSH tunnel then would be

ssh -f -C bilbo@pop.foobar.com -L 1234:pop.foobar.com:110 sleep 5

(For testing purposes, you might want to increase the value for *sleep* to 500). This should prompt you for your POP password

bilbo@pop.foobar.com's password:

Having provided your password, use *telnet* to connect to the local forwarded port

telnet localhost 1234

You now should get a READY message from the remote mail server.

Of course this method would require you to type in all POP commands by hand, which might be a bit - inconvenient ;-). To automate this, use Fetchmail (see this page on how to configure Fetchmail, if it isn't running on your box already). The Secure POP via SSH mini-HOWTO, man fetchmail and the Fetchmail FAQ in */usr/doc/fetchmail-[...]/* all feature example configurations.

Note that the IMAP protocol uses different ports: IMAP v2 uses port 143 and IMAP v3 uses port 220.

F. Interfaz de AGW

G. Uso de PQDI

H. Tutorial de EDLIN

I. Edición del registro de Windows

J. Scripts de inicialización de módems en Linux

K. Generador de paquetes de instalación Windows

Para la instalación del *daemon* cliente para Windows se ha hecho uso de un programa que automatiza todo el proceso, la instalación de ficheros, los accesos directos y la modificación del registro. El programa escogido es **Setup Generator** [18], gratuito y de uso libre.

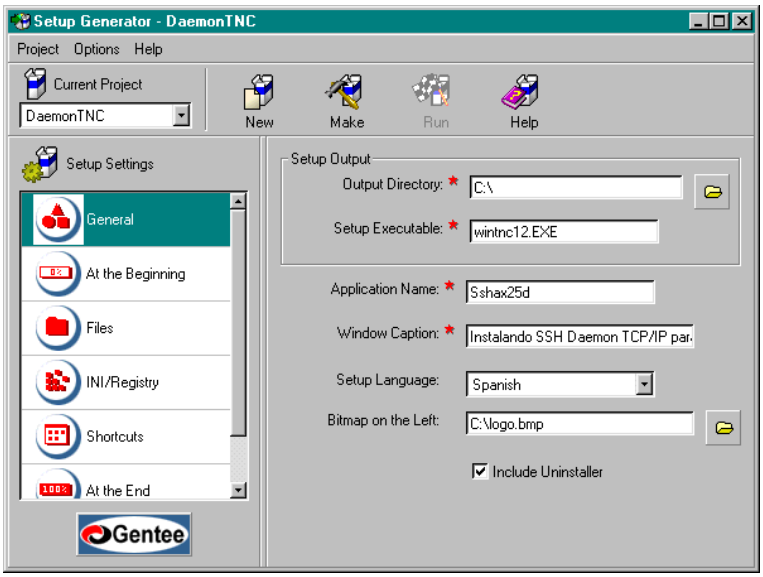
K.1. Configuración

Una vez instalado el programa lo ejecutamos y creamos un nuevo proyecto:

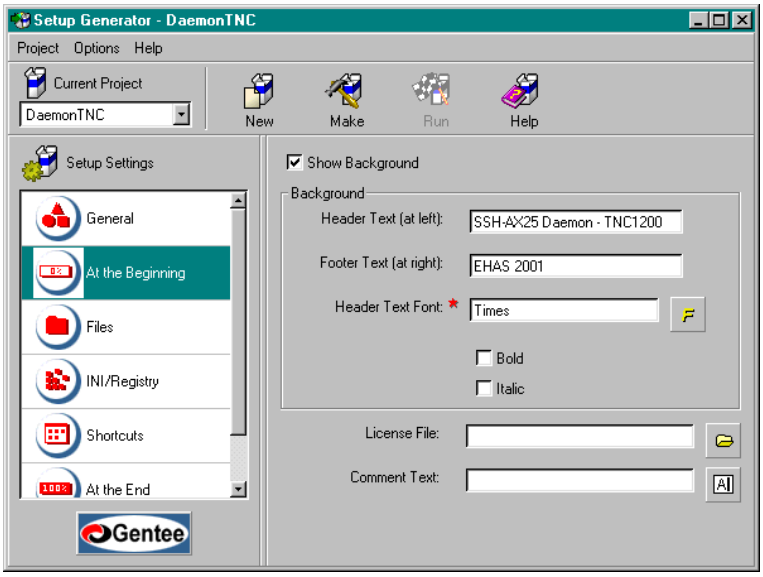
Project →New.

Ahora hay que completar la configuración del proyecto que hemos creado:

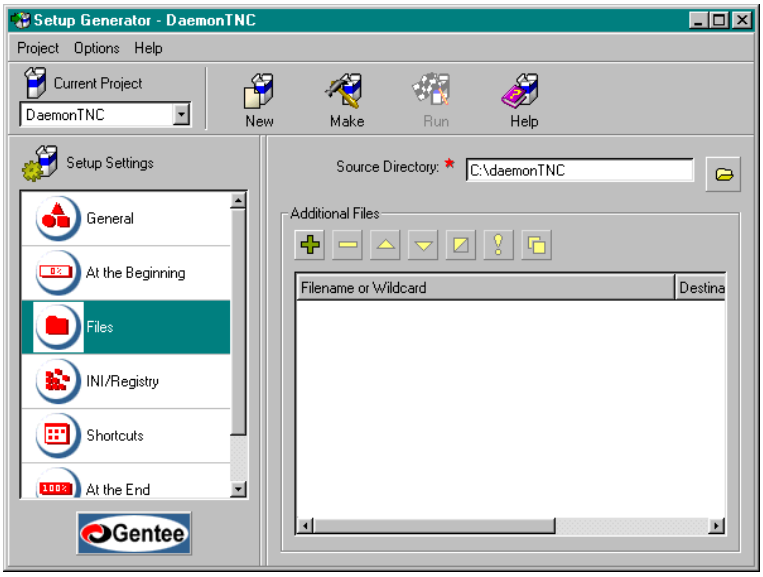
General. En este apartado definimos el nombre de la aplicación, el nombre del ejecutable a crear y especialmente el idioma en el que se creará (seleccionar `Spanish`).



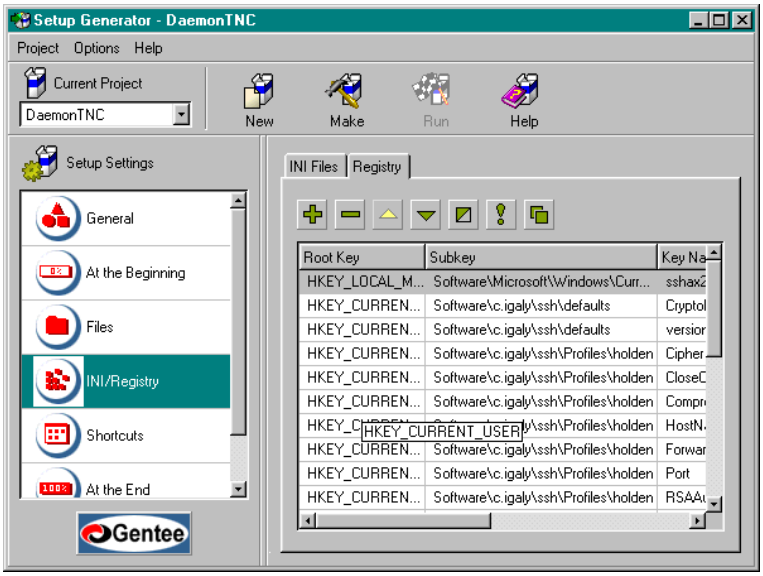
At the Beginning. Ponemos el título de la ventana y el tipo de letra.



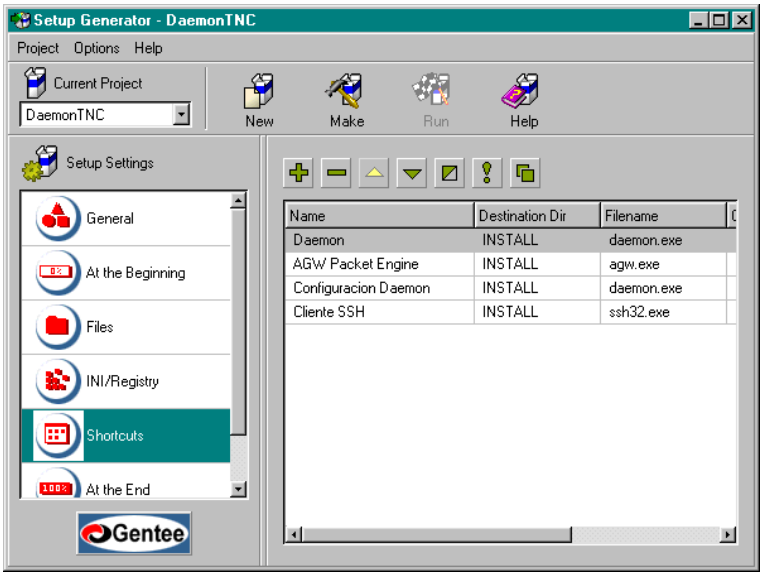
Files. Ponemos como *Source directory* el directorio que contenga los ficheros que integran el paquete. En este caso `C:\daemonTNC`



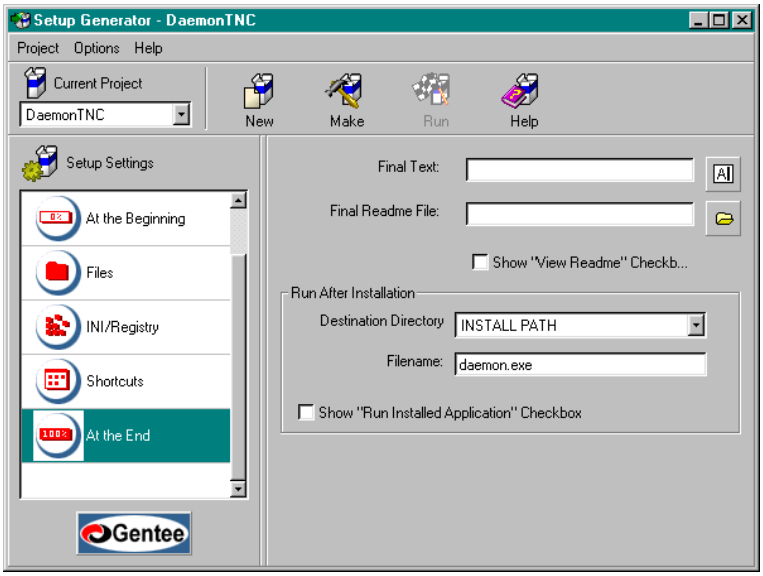
INI/Registry. En este apartado se configura las modificaciones en el registro que se deseen hacer. Todos las modificaciones hacen referencia al cliente SSH32, configurando los puertos redireccionados.



Shortcuts. Indicamos los accesos directos que queremos crear en la carpeta que se creará en el menú de Programas.



At the End. Tenemos la posibilidad de ejectuar un programa una vez finalizada la instalación. En nuestro proyecto nos interesa que se ejecute el *daemon* para que, como el fichero de configuración no existe aún, se abra la ventana de configuración.



Ahora ya podemos hacer **Make** y se generará el ejecutable en el directorio destino seleccionado. Una limitación que tiene el programa **Setup Generator** es que no es capaz de generar accesos directos en el menú **Inicio** ni tampoco crear entradas en el registro para que un programa se ejecute en el arranque. Por ello es el propio *daemon* quien al iniciarse por primera vez escribe en el registro de Windows para asegurar que se inicie en el arranque tal como se vio en la sección 6.6.

L. Licencia Pública General GNU

Versión 2, junio de 1991

Copyright (C) 1989, 1991 Free Software Foundation, Inc. 675 Mass Ave, Cambridge, MA 02139, USA Se permite a todo el mundo copiar y distribuir copias idénticas de este documento de licencia, aunque no se permite su modificación.

L.1. Preámbulo

Las licencias de uso de casi todo el software han sido establecidas para quitarle la libertad de compartirlo y modificarlo. En contraste con esta costumbre, la Licencia General Pública GNU pretende garantizar su libertad de compartir y modificar software libre, a fin de asegurar que el software es libre para todos sus usuarios. Esta Licencia General Pública es de aplicación para la mayor parte del software de la Free Software Foundation y para cualquier otro programa cuyos autores se comprometen a su utilización. (Otro software de la Free Software Foundation está cubierto alternativamente por la Licencia General Pública de biblioteca GNU). También puede aplicar ésta a sus programas. Cuando hablamos de software libre nos referimos a la libertad, no al precio. Nuestras Licencias Generales Públicas han sido diseñadas para asegurarle la libertad necesaria para distribuir copias de software libre (y de cobrar por este servicio, si así lo desea), que reciba código fuente o pueda obtenerlo si lo desea, que pueda modificar el software o utilizar partes del mismo en nuevos programas libres y que sepa que puede hacer estas cosas. A fin de proteger sus derechos, es necesario constituir restricciones que impidan que alguien pueda denegarle estos derechos o pedirle que renuncie a los mismos. Estas restricciones implican también ciertas responsabilidades si distribuye copias del software o lo modifica. Por ejemplo, si distribuye copias de este programa, tanto gratuitamente o con ánimo de lucro, deber conceder a los recipientes del mismo todos los derechos con los que cuente. Deberá asegurarse de que los recipientes reciben o pueden obtener el código fuente. Y deber asimismo mostrarle estos términos, a fin de que puedan conocer sus derechos. Protegemos sus derechos con dos pasos: (1) copyright del software y (2) le ofrecemos esta licencia que le concede permiso legal para copiar, distribuir y/o modificar el software. Además, para la protección de cada autor así como la nuestra propia, deseamos asegurarnos que todo el mundo comprende que este software libre no cuenta con garantía alguna. Si el software es modificado por alguna otra persona y posteriormente distribuido, deseamos que sus destinatarios sepan que lo que reciben no es el producto original, de forma que cualquier problema introducido por otras partes no pueda reflejarse en la reputación profesional de los autores originales. Finalmente, cualquier programa libre se ve constantemente amenazado por patentes de software. Deseamos evitar el peligro de que los redistribuidores de un programa libre obtengan licencias de patente a título individual, lo que haría de éste un programa privado. A fin de impedir esto hemos dejado muy claro que cualquier patente obtenida deberá ser licenciada para la utilización libre de todos o no deberá ser licenciada en absoluto. A continuación se encuentran los términos y condiciones precisos para la copia, distribución y modificación.

L.2. Términos y condiciones para la realización de copias, distribución y modificación

1. Esta Licencia es de aplicación a cualquier programa u otro trabajo que contenga un aviso incluido por el titular de los derechos de copyright estableciendo su libertad de distribución bajo los términos de esta Licencia General Pública. El "Programa", a continuación, hace referencia a cualquier programa o trabajo de este tipo y un "trabajo basado en el Programa"indica el Programa o cualquier tipo de trabajo derivado bajo las leyes de copyright: es decir, un trabajo que contenga el Programa o una porción del mismo, bien en versión exacta o bien con modificaciones y/o traducido a otro idioma. (En adelante, las traducciones quedan incluidas, sin limitación alguna, en el término "modificación"). Cada beneficiario de esta licencia quedar á indicado por el término üsted". Las actividades que no incluyan copia, distribución y modificación no quedarán protegidas por esta Licencia; quedan fuera de su alcance. El acto de ejecución del Programa no queda limitado y los resultados obtenidos de la ejecución del Programa quedan cubiertos sólo si su contenido constituye un trabajo basado en el Programa (independientemente de que haya sido obtenido mediante la ejecución del Programa). La validez de esto dependerá de lo que haga el Programa.

2. Podrá copiar y distribuir copias exactas del código fuente del programa según lo hubiera recibido, utilizando cualquier medio, siempre que de forma notoria y apropiada publique, en cada copia, un aviso adecuado de derechos de copyright y relativo a la ausencia de garantía alguna; y de que proporcione a todos los destinatarios del Programa una copia de esta Licencia junto con el mismo. Podrá cobrar honorarios por el acto físico de transferencia de una copia y podrá, a su propia discreción, ofrecer servicios de protección de garantía a cambio de honorarios.

3. Puede modificar su copia o copias del Programa o cualquier porción del mismo, hasta la formación de un trabajo basado en el Programa, y podrá asimismo copiar y distribuir dichas modificaciones o trabajos bajo los términos de la Sección 1 anteriormente reseñada, siempre que cumpla las condiciones adicionales que se detallan a continuación:

   a) Deberá asegurarse de que los archivos modificados incorporen avisos prominentes que establezcan que ha modificado los archivos, así como la fecha de dicha modificación.

   b) Deberá asegurarse de que cualquier trabajo que distribuya o publique, que pudiera, en su totalidad o en parte, contener o haber sido derivado del Programa o de cualquier parte del mismo, sea licenciado en su totalidad y sin cargo alguno a terceros que lo requieran bajo los términos de esta Licencia.

   c) Si el programa modificado lee normalmente comandos interactivamente cuando se ejecuta, deberá asegurarse de que, cuando se ponga en marcha su ejecución para dicha utilización interactiva de la forma más común, imprima o muestre una proclama que incluya un aviso correspondiente relativo a los derechos de copyright y una clarificación de la falta de garantía (o estableciendo la provisión de dicha garantía por su parte) y que confirme que los usuarios pueden redistribuir el programa bajo estas condiciones, e informe al usuario cómo visualizar una copia de esta Licencia. (Excepción: si el Programa es de por si interactivo pero no imprime normalmente dicha proclama, tampoco se requerirá que su trabajo basado en el Programa deba imprimirla). Estos requisitos son de aplicación al trabajo modificado en su totalidad. Si hubiera secciones identificables de dicho trabajo no derivadas del Programa y que pudieran considerarse trabajos razonablemente independientes, entonces esta Licencia, así como sus términos, no serian de aplicación a dichas

secciones cuando las distribuya como trabajos independientes. Pero cuando distribuya las mismas secciones como parte de un trabajo completo mayor basado en el Programa, la distribución del trabajo completo deberá realizarse bajo los términos de esta Licencia, cuyos permisos con respecto a otras licencias se ampliarán a la totalidad del trabajo completo y por lo tanto a cada una de las partes del trabajo irrespectivamente de quién lo ha escrito. Por todo esto, no es el propósito de esta sección reclamar derechos o disputar sus derechos a trabajos escritos enteramente por el titular de esta licencia sino ejercitar el derecho a controlar la distribución de trabajos derivados o colectivos basados en el Programa. Además, la simple agregación de otro trabajo no basado en el Programa a éste (o con un trabajo basado en el Programa) en un volumen de un medio de almacenamiento o distribución no somete al otro trabajo al alcance de esta Licencia.

4. Podrá copiar y distribuir el Programa (o un trabajo basado en él mismo, bajo los términos de la Sección 2) en código objeto o formato ejecutable bajo los términos anteriormente reseñados en las Secciones 1 y 2, siempre que observe asimismo una de las condiciones siguientes:

   a) Lo acompañe con el correspondiente código completo leíble por él sistema, que deber á distribuirse bajos los términos anteriormente reseñados en las Secciones 1 y 2 en un medio normalmente utilizado para intercambios de software; o

   b) Lo acompañe con una oferta por escrito, con una validez mínima de tres años. de facilitar a terceros, a un precio no superior al coste de realizar la distribución física del código, una copia completa leíble por la máquina del correspondiente código fuente, a distribuir bajo los términos anteriormente reseñados en las Secciones 1 y 2 en un medio normalmente utilizado para intercambios de software; o

   c) Lo acompañe con la información que haya recibido en relación a la oferta de distribución del código fuente. (Esta alternativa se permite exclusivamente para distribuciones no comerciales y sólo si ha recibido el programa en código objeto o formato ejecutable con dicha oferta, de acuerdo con los términos anteriormente reseñados en la Subsección b). El código fuente de un trabajo significa la forma preferida del trabajo para la realización de modificaciones al mismo. Para un trabajo ejecutable, el código fuente completo significa todo el código fuente para todos los módulos que contiene, más cualesquiera archivos de definición de interfaz asociados, más las secuencias que se utilicen para controlar la compilación y ejecución del ejecutable. Sin embargo, como una excepción especial, el código fuente distribuido no necesita incluir nada que sea normalmente distribuido (en formato fuente o binario) con los componentes principales (compilador, núcleo, etc.) del sistema operativo en el que se ejecuta el ejecutable, a menos que el propio componente acompañe al ejecutable. Si llegara a realizarse una distribución de código objeto o ejecutable ofreciendo acceso a copiar de un lugar designado, el ofrecimiento de acceso equivalente a copiar el código fuente del mismo lugar contará como una distribución del código fuente, incluso cuando las terceras partes no tengan que copiar el código fuente conjuntamente con el código objeto.

5. No podrá copiar, modificar, sublicenciar o distribuir el Programa salvo en la forma en que esta Licencia expresamente lo permita. Cualquier otro intento de copia, modificación, sublicencia o distribución del Programa será nula y dará fin de forma automática a sus derechos bajo los términos de esta Licencia. Sin embargo, las licencias correspondientes a terceros que hubieran podido recibir copias, o derechos, del titular de esta licencia bajo los términos de la misma no quedarán canceladas siempre que dichas partes observen sus condiciones.

6. No tiene obligación alguna de aceptar esta Licencia, ya que no ha firmado la misma. Sin embargo, nada aquí le concede permiso de modificación o distribución del Programa o sus trabajos derivados. Estas acciones quedarán prohibidas por la ley si decidiera no aceptar esta Licencia. Por lo tanto, mediante la modificación o distribución del Programa (o de cualquier trabajo basado en el mismo), indicará su aceptación de esta Licencia y de sus términos y condiciones para la copia, distribución o modificación del Programa o de los trabajos basados en el mismo.

7. Cada vez que redistribuya el Programa (o cualquier trabajo basado en el mismo), el destinatario recibe automáticamente una licencia del titular original para copiar, distribuir o modificar el Programa en base a estos términos y condiciones. No podrá imponer restricción adicional alguna sobre el ejercicio, por parte del destinatario, de los derechos que aquí se conceden. No tendrá responsabilidad alguna en lo que atañe a asegurar la observación, por terceros, de las condiciones a las que está sujeta esta licencia.

8. Si, como consecuencia de una sentencia judicial de infracción de derechos de patente o por cualquier otra razón (no limitada a cuestiones de patente), se le impusiera condición alguna (por sentencia judicial, acuerdo, o cualquier otra vía) que estuviera en contradicción con las condiciones de esta Licencia, dicha condición no podrá constituir excusa en lo que atañe a la obligación de observar las condiciones de esta Licencia. Si no pudiera efectuar distribuciones de forma que satisfaga simultáneamente sus obligaciones bajo los términos de esta Licencia y cualesquiera otras obligaciones que le pudieran corresponder, la consecuencia de dichas circunstancias es que no podrá distribuir el Programa en absoluto. Por ejemplo, si una licencia de patente no permitiera la redistribución libre de derechos de royalty del Programa por parte de aquellos que recibieran copias de forma directa o indirecta a través suya, entonces la única forma en que podría satisfacer dicha licencia de patente y esta Licencia consistiría en no distribuir el Programa en absoluto. Si cualquier porción de esta sección llegara a ser declarada inválida o no ejecutable bajo cualquier circunstancia determinada, el resto de la sección seguirá siendo de aplicación y la sección completa seguirá siendo asimismo aplicable siempre que concurran otras circunstancias. No es el propósito de esta sección inducirle a infringir patentes u otros derechos de la propiedad intelectual o disputar en forma alguna la validez de dichos derechos; el único objetivo de esta sección es la protección de la integridad del sistema de distribución de software libre, implantado por prácticas de licencia pública. Muchas personas han hecho generosas contribuciones a una amplia gama de software distribuido a través de este sistema confiando en la coherente aplicación de este sistema de distribución; depende del autor/donante decidir si desea distribuir el software mediante cualquier otro sistema, y un licenciado no puede establecer imposición alguna con respecto a dicha decisión. Esta sección pretende aclarar perfectamente lo que se supone es una consecuencia del resto de esta licencia.

9. Si la distribución y/o utilización del Programa quedara restringida en ciertos países bajo el efecto de derechos de patente o interfaces sujetas a copyright, el titular original de los derechos de copyright que ampara el Programa bajo esta Licencia podrá añadir una limitación explícita del ámbito de distribución geográfica que excluya a dichos países, de forma que la distribución quede permitida sólo en países, o entre países, que no estén sujetos a dicha exclusión. En este caso, esta Licencia incorporar la limitación de la misma forma que si hubiera sido incluida en el texto de esta Licencia.

10. La Free Software Foundation podrá publicar versiones revisadas y/o nuevas de la Licencia General Pública de forma periódica. Estas nuevas versiones serán similares en espíritu a la versión actual. aunque podrían diferir en detalles a fin de resolver nuevos problemas o cuestiones. Cada versión recibirá un número de versión distinto. Si el Programa especifica un número de versión de esta Licencia que sea de aplicación a la misma y a çualquier

otra versión posterior", tendrá la opción de seguir los términos y condiciones de dicha versión o de cualquier versión posterior publicada por la Free Software Foundation. Si el Programa no especificara un número de versión de esta Licencia, podrá seleccionar cualquier versión publicada en cualquier momento por la Free Software Foundation.

11. Si deseara incorporar partes del Programa en otros programas libres cuyas condiciones de distribución sean distintas, escriba al autor y solicite permiso. Escriba a la Free Software Foundation con respecto al software cuyos derechos de copyright correspondan a esta organización; algunas veces concedemos excepciones a estas normas. Nuestra decisión será guiada por los dos objetivos de conservar el carácter libre de todos los derivados de nuestro software libre y de promover generalmente la participación en el software y su reutilización. INEXISTENCIA DE GARANTIA

12. DEBIDO A QUE EL PROGRAMA SE LICENCIA LIBRE DE COSTE ALGUNO, EN LO QUE PERMITE LA LEY, ESTE PROGRAMA NO ESTA PROTEGIDO POR GARANTÍA ALGUNA. SALVO CUANDO SE ESTABLEZCA LO CONTRARIO POR ESCRITO, LOS TITULARES DE LOS DERECHOS DE COPYRIGHT Y/O OTRAS PARTES SUMINISTRAN EL PROGRAMA "TAL Y COMO ES", SIN GARANTÍA DE NINGUN TIPO, EXPRESA O IMPLÍCITA, INCLUYENDO, PERO SIN QUE ESTO SIRVA DE LIMITACIÓN, LAS GARANTÍAS IMPLÍCITAS DE COMERCIABILIDAD O UTILIDAD PARA ALGÚN PROPÓSITO DETERMINADO. LA TOTALIDAD DEL RIESGO EN RELACION A LA CALIDAD Y RENDIMIENTO DEL PROGRAMA DESCANSA EN EL USUARIO. SI SE LLEGARA A ENCONTRAR QUE EL PROGRAMA ADOLECE DE ALGUN DEFECTO, EL USUARIO DEBERA ASUMIR TODOS LOS COSTES QUE RESULTARAN NECESARIOS PARA SU SERVICIO, REPARACION O CORRECCION.

13. BAJO NINGUNA CIRCUNSTANCIA, A MENOS QUE LO REQUIERA ALGUNA LEY APLICABLE O QUE SE ACUERDE POR ESCRITO, EL TITULAR DE LOS DERECHOS DE COPYRIGHT O CUALQUIER OTRA PARTE QUE HUBIERA PODIDO MODIFICAR Y/O REDISTRIBUIR EL PROGRAMA DE ACUERDO CON LAS CONDICIONES ANTERIORMENTE RESEÑADAS, SERÁ RESPONSABLE ANTE EL USUARIO POR DAÑOS Y PERJUICIOS, INCLUYENDO DAÑOS Y PERJUICIOS GENERALES, ESPECIALES, INCIDENTALES O CONSECUENTES, RESULTANTES DE LA UTILIZACIÓN O INCAPACIDAD DE USO DEL PROGRAMA (INCLUYENDO, PERO SIN QUE ESTO CONSTITUYA UNA LIMITACIÓN, LA PÉRDIDA DE DATOS O LA INEXACTITUD DE DICHOS DATOS O LAS PÉRDIDAS SOSTENIDAS POR EL USUARIO O TERCEROS O UN FALLO DEL PROGRAMA CON RESPECTO A SU FUNCIONAMIENTO CON OTROS PROGRAMAS), INCLUSO SI DICHO TITULAR U OTRA PARTE HUBIERA SIDO INFORMADO CON RESPECTO A LA POSIBILIDAD DE DICHOS DAÑOS Y PERJUICIOS. FIN DE LOS TÉRMINOS Y CONDICIONES

L.3. Apéndice

Como Aplicar estos Términos a sus Nuevos Programas Si llegara a desarrollar un nuevo programa y desea fuera de la mayor utilidad posible al público, la mejor forma de conseguirlo es escribir software libre que todo el mundo pueda redistribuir y cambiar libremente bajo estos términos. Para hacer esto, adjunte los avisos que se relacionan a continuación al programa. Lo más seguro es adjuntarlos al principio de cada archivo fuente a fin de expresar con la mayor efectividad la exclusión de garantías; y cada archivo debería tener por lo menos la línea de

çopyrightz un indicador que informe sobre dónde puede localizarse el texto completo. Úna línea para el nombre del programa y una pequeña idea de lo que hace¿Copyright © l9yy ¡nombre del autor¿Este programa es software libre; lo puede reditribuir y/o modificar bajo los términos de la Licencia General Pública GNU según lo publicado por la Free Software Foumdation, bien la versión 2 de la Licencia o (a su propia opción) cualquier otra versión posterior. Este programa se distribuye con la esperanza de que será útil, pero SIN NINGÚN TIPO DE GARANTÍA; incluso sin la garantía implícita de COMERCIABILIDAD o UTILIDAD PARA ALGÚN PROPÓSITO DETERMINADO. Mire la Licencia pública general de GNU para más información. Debería haber recibido una copia de la Licencia General Pública GNU junto con este programa; en caso contrario, escriba a la Free Software Foundation, Inc., 675 Mass Ave, Cambridge MA 02139 USA Adjunte también información sobre cómo contactar por correo electrónico o por correo normal. Si el programa es interactivo, haga que proporcione un corto aviso como el que se muestra a continuación cuando comienza en modalidad interactiva: Gnomovision version 69, Copyright © 19yy name of author Gnomovision comes with ABSOLUTELY NO WARRANTY;for details type 'show w' This free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details. Los comandos hipotéticos 'show w' y 'show c' deberían mostrar las secciones correspondientes de la Licencia General Pública. Como es natural, los comandos que utilice podrían denominar algo diferente a 'show w' y 'show c', podrían incluso ser pulsaciones de los botones del ratón o elementos de menú, lo que se adecue más a su programa. También debería hacer, si lo creyera necesario, que su empresa (si trabaja como un programador) o su institución educativa, si correspondiera, firmara un .ªviso de no aplicabilidad de derechos de copyright". A continuación se muestra un ejemplo, modifique los nombres de acuerdo con sus requisitos: Yoyodyne, Inc., por este medio renuncia a todos los derechos de copyright del programa "Gnomovision"(which makes passes at compilers) escrito por James Hacker. <firma del Empresario>, 1 Abril 1989 Empresario, o Director en su lugar

Esta Licencia General Pública no permite la incorporación de su programa a programas privados. Si su programa es una biblioteca de subrutinas, podría considerar más útil permitir el enlace de aplicaciones privadas con la biblioteca. Si esto es lo que desea hacer, utilice la Licencia General Pública de Biblioteca GNU en lugar de esta Licencia.

Pliego de condiciones

Condiciones generales

Artículo 1: La ejecución de la instalación se llevará a cabo por el procedimiento de contratación directa.

Artículo 2: Las relaciones laborales y administrativas entre el contratista y el contratado, a efectos del desarrollo del proyecto, estarán sujetas a la normativa vigente. El adjudicatario está igualmente obligado al cumplimiento de toda la legislación vigente sobre seguridad e higiene en el trabajo y fomento del consumo de artículos nacionales.

Artículo 3: El contratista tiene derecho a obtener, a su costa, cuantas copias requiera de los planos, pliego de condiciones y presupuesto, copias que serán autorizadas por el ingeniero director con su firma después de ser confrontadas con el original.

Artículo 4: Las cargas y salarios deberán ajustarse al presupuesto en cuanto a cantidad y duración. Para cualquier variación, el contratista deberá pedir la debida autorización al ingeniero proyectista.

Artículo 5: Se abonará al contratista la obra que realmente ejecute de acuerdo con el proyecto que sirva de base para el contrato, sin que su importe pueda exceder en ningún caso de la cifra total de los presupuestos aprobados.

Artículo 6: Si se juzgara necesario efectuar modificaciones que no figuren en el presupuesto de la contrata se valorará su importe, de acuerdo a los precios asignados a estas obras y otras análogas, si las hubiera.

Artículo 7: El contratista u órgano correspondiente queda obligado a abonar al ingeniero autor del proyecto y director de los trabajos, así como a sus respectivos ayudantes, el importe de sus respectivos honorarios facultativos por la realización del proyecto, realización técnica y administrativa, de acuerdo con las tarifas y honorarios vigentes.

Artículo 8: En los gastos generales se incluyen los correspondientes a mecanografiado, así como los de amortización y cargas fiscales.

Artículo 9: El software desarrollado, propiedad intelectual del ingeniero proyectista, se distribuirá bajo licencia GPL (en el CD en el que se distribuirá el software desarrollado ha sido incluida una copia de esta licencia). Esta licencia deberá incluirse en cualquier copia parcial o completa que se haga del software desarrollado.

Artículo 10: El contratista, por el hecho de contratar el proyecto, se compromete a aceptar todas las claúsulas de este pliego de condiciones, los planos y el presupuesto adjunto.

Condiciones técnicas

Artículo 1 La instalación y configuración del servidor y los clientes deberá realizarse con los programas y procedimientos descritos en los anexos

Presupuesto

Presupuesto de Ejecución Material

Materiales	
2 Ordenadores portátiles Airis H-2122	144.900
2 Transceptores Yaesu VX-2000	119.500
2 Transceptores Motorola PRO3100	152.000
2 Cargas (*Dummy Load*) D21M (50 Ohms)	11.000
Componentes montaje placa PTT de SB	2.000
2 TNCplus 9600	55.800
2 Módems PicPar	29.000
2 Módems YAM	32.000
Fuente de alimentación 40XII (20V - 40A)	38.500
Total Materiales	584.700

Honorarios del Ingeniero Proyectista

Conceptos

Salario del Ingeniero de Telecomunicación. Contempla el trabajo realizado por un Ingeniero de Telecomunicación durante 6 meses, a razón de cinco días por semana y ocho horas diarias con el siguiente desglose:

Especificación de requisitos y diseño. 15 días

Estudio de módems y radios. 30 días

Estudio de aplicaciones auxiliares de Windows. 15 días

***Daemon* servidor** 15 días

***Daemon* cliente** 15 días

Implementación de DAMA 30 días

Gestión remota 30 días

Redacción y documentación 30 días

Número de horas de desarrollo = 8 x 5 x 4 x 6 = 960 horas
Coste = 960 x 8000 ptas./hora = 7.680.000 ptas.

Coste salario del Ingeniero de Telecomunicación 7.680.000

Coste de redacción. El coste de redacción engloba el coste de mecanografiado y el coste de impresión y encuadernación. El coste de mecanografiado es de 1.000.000 ptas. (10 semanas, 5 días por semana, 8 horas diarias y 2.500 ptas./hora), mientas que el coste de impresión y de encuadernación se ha estimado en 40.000 ptas. Al final, el coste de redacción asciende a 1.040.000 ptas.

Coste de redacción 1.040.000

Hoja de Minuta de Honorarios

Minuta de Honorarios	
Salario del Ingeniero de Telecomunicación	7.680.000
Coste de redacción	1.040.000
Subtotal	8.720.000
IVA 16 %	1.228.800
Total	8.948.800

Presupuesto de Contrata

Materiales	584.700
Gastos Generales 6 %	35.082
Beneficio Industrial 16 %	93.552
Presupuesto de Contrata	713.334

Presupuesto Total del Proyecto

Los honorarios del Ingeniero Director representan el 7 % del Presupuesto de Ejecución Material.

Presupuesto de Contrata	713.334
Honorarios del Ingeniero Proyectista	9.680.000
Subtotal	10.393.334
IVA 16 %	1.662.933
Importe Total del Proyecto	12.056.267
	72.459, 62 Euros

El Importe Total del Proyecto asciende a doce millones, cincuenta y seis mil dos cientas sesenta y siete pesetas (12.056.267 ptas).

Madrid, a 20 de septiembre de 2001

El Ingeniero de Telecomunicación,

Arnau Sánchez Sala