Capítulo 3

Desarrollo del trabajo

En este capítulo se abordan las cuatro partes de las que consta el proyecto: el control de acceso, el reparto de ancho de banda, las políticas de acceso y la administración del sistema. En cada una de las partes se ha intentado dar una visión general del problema, las necesidades del proyecto y después se ha explicado la solución adoptada.

3.1. Control de acceso

El sistema planteado por NoCat está bastante bien pero tiene ciertos inconvenientes. En primer lugar no hay control de estado en el servidor de autenticación. Simplemente se utilizan dos tablas para comprobar si el usuario existe o no. Es más, con el sistema de NoCat es posible que varios usuarios distintos utilicen el mismo login para acceder a Internet simultáneamente. Esto se debe a que el servidor no sabe quien está conectado y quien no, simplemente recibe peticiones, comprueba sus tablas y devuelve las credenciales.

Nosotros pretendemos que, además de permitir o no el uso de Internet de una forma estática, se lleve un control de quién se conecta, cuándo se conecta y cuánto tiempo se conecta. De esta manera se podrá cobrar a los usuarios por el servicio recibido. Además de controlar las conexiones, también pretendemos tener capacidad para restringir el acceso a un usuario cuando se agote el servicio contratado.

3.1.1. Sistema de tarificación

Cuando un usuario pretenda navegar por Internet a través de nuestro sistema deberá contratar un servicio, que en primer lugar definirá un tipo de tarifación para ese usuario. Los tipos de tarifación que se han contemplado son tres:

- Tarifa Plana. La tarifa plana establece que un usuario podrá navegar desde un instante hasta otro instante prefijados. Los instantes se definen mediante una fecha y una hora, de manera que la tarifa plana puede ser diaria, mensual, o simplemente durante unas horas (lo que se llamaría en este caso prepago). Cuando pasa el instante de baja no se debe permitir al usuario seguir navegando.

- Tarifa Pospago. Este usuario tiene un instante de alta pero no de baja. Puede navegar cuanto quiera y no se le corta nunca la conexión. En el momento de facturación debemos saber cuánto tiempo ha estado conectado.

- Tarifa Bono. Un usuario puede comprar un bono de un tiempo cualquiera (HORAS:MINUTOS:SEGUNDOS) que irá agotando cuando esté conectado. En el momento que se agote el bono a este usuario, no se le permitirá que siga conectado.

3.1.2. Solución adoptada

Para llevar a cabo el control de acceso se ha intentado hacer un sistema independiente al sistema Nocat de forma que cuando salgan versiones posteriores de Nocat sigan siendo compatibles con nuestra aplicación.

La idea se basa en que Nocat permite el acceso a Internet a los usuarios que están en la tabla *member*. Si nuestra aplicación inserta en esta tabla a los usuarios en los periodos que están contratados, y los borra cuando se acaba el contrato, ya hemos conseguido lo que queríamos y apenas hay que cambiar el sistema Nocat.

Con este propósito nosotros creamos una nueva base de datos llamada ControlNocat, en esta base de datos se incluyen las tablas necesarias para controlar la activación de los usuarios y sus conexiones.

Tablas de la base de datos ControlNocat

Se utilizarán cuatro tablas: activación, histórico de activación, conexiones e histórico de conexiones.

Activación

En *activacion*[1] se guarda la información relacionada con los usuarios que tienen contratado un servicio. Los campos que incorpora y el tipo dentro de MySQL son:

[1]Se ignorará la tilde de activacion porque su nombre real dentro de la base de datos no tiene tilde.

login	VARCHAR(8) PRIMARY KEY
tarifacion	ENUM("pdiaria","pmensual","pospago","bono","prepago")
grupo	VARCHAR(20)
alta	DATETIME
baja	DATETIME
bono	TIME
tarifa	FLOAT(4)
facturado	ENUM("si","no")
ubicacion	VARCHAR(17)

La función de cada campo es:

- El campo login será la clave primaria de esta tabla.
- El campo tarifación se ha elegido como tipo ENUM y a la hora de insertar los valores en la tabla se puede hacer tanto con su nombre como con el lugar que ocupa en la lista, por ejemplo al tipo pospago le corresponde el tres.
- El campo grupo nos permite conocer el grupo al que pertenece el usuario, dentro de la tabla *grupo* se guardan los parámetros de conexión de un grupo. La tabla de grupo la veremos más adelante.
- Los campos alta y baja son DATETIME y tienen el formato AAAA-MM-DD HH:MM:SS. El campo alta se rellena siempre, mientras que el campo baja sólo se rellena en las tarifas planas(pdiaria, pmensual y prepago). Cuando el tipo es bono o pospago no tiene fecha límite impuesta de antemano.
- El campo bono es de tipo TIME, tiene formato HH:MM:SS, e indica el tiempo que le queda por consumir al bono. Este campo se actualiza cada vez que ese usuario se conecta y se va reduciendo el tiempo que le queda. Evidentemente este campo sólo se rellena cuando el usuario es de tipo bono. Para el resto de tipos de tarifación el valor es NULL.
- El campo tarifa contiene la tarifa de ese usuario.
- El campo facturado sirve para saber si ya se ha facturado o no, esa activación. Es un campo lógico, pero MySQL no tiene tipo binario, por lo que hemos optado por un tipo enum con los valores 'si' y 'no'.
- En el campo ubicacion se guarda la MAC de la pasarela desde la cual se puede conectar ese usuario. Si intenta conectarse desde otra pasarela el servidor de autenticación rechazará la petición. Si este campo tiene valor "NULL" quiere decir que el usuario puede conectarse desde cualquier ubicación.

Conexiones

En *conexiones* se guardan temporalmente las conexiones que están activas. Los campos que se han definido y sus tipos correspondientes en MySQL son:

login	VARCHAR(8) PRIMARY KEY
tarifacion	ENUM("pdiaria","pmensual","pospago","bono","prepago")
inicio	DATETIME
fin	DATETIME
id_cola	VARCHAR(4)
ubicacion	VARCHAR(17)
tarifa	FLOAT(4)
facturado	ENUM("si","no")

La función de cada campo es:

- Al igual que antes la clave primaria de esta tabla también es login.
- En el campo tarifación se guarda el tipo de conexión que es.
- El campo inicio se rellena para todas las conexiones, con el instante que empieza la conexión.
- El campo fin se rellena para las conexiones que tienen un instante de fin, que son todas excepto las de tipo pospago en las que se deja con valor NULL.
- El campo id_cola se verá más adelante en el apartado de calidad de servicio.
- El campo ubicacion se rellena con la dirección MAC de la pasarela desde la que está conectado el usuario.
- La tarifa es el precio al que se le va a cobrar esta conexión al usuario en caso de que sea pospago.
- Facturado tiene el mismo significado que en la tabla *activacion*.

Histórico de Activación

En *hco_activacion*[2] se guarda un histórico de los valores que se insertan en la tabla *activacion*. Esta tabla nos sirve tanto para facturar como para llevar una estadística de los servicios que se contratan. Los campos que incorpora y el tipo dentro de MySQL son:

[2]Se ignorará la tilde de hco_activacion porque su nombre real dentro de la base de datos no tiene tilde.

cod	INT(11) auto_increment
login	VARCHAR(8)
tarifacion	ENUM("pdiaria","pmensual","pospago","bono","prepago")
grupo	VARCHAR(20)
alta	DATETIME
baja	DATETIME
bono	TIME
tarifa	FLOAT(4)
facturado	ENUM("si","no")
ubicacion	VARCHAR(17)

El modo de rellenar esta tabla no es lo normal a la hora de trabajar con históricos. No se va a guardar en esta tabla cada vez que cambie la activación, ni cuando se de de baja el usuario. En realidad esta tabla va a contener el tiempo que realmente está activado el usuario. Es decir, registra los momentos en los que se le permite a un usuario navegar. A la hora de facturar a los usuarios de tarifa plana se utilizará esta tabla para ver los periodos activos de cada usuario. Puede existir usuarios que se den de alta para un fecha posterior a la actual, y después se cambien o se den de baja, estas entradas en la tabla activación no se registrarán en el histórico, ya que no se han hecho efectivas.

El modo de actuar será el siguiente:

1. Cuando se da de alta un usuario se crea una fila en *activacion* con los datos de activación del usuario.

2. En el caso de que se esté dentro del periodo de activación, se inserta la misma fila en el histórico de activación pero permanece el campo de baja igual a 'NULL'.

3. Cuando una fila de *activacion* cambia, se busca en el histórico una fila con el mismo login y el campo de baja en 'NULL'. En esta fila, se pone en el campo de baja el instante actual del cambio. Si después del cambio, el usuario sigue activo, se creará una nueva fila con los valores nuevos en *hco_activacion* y otra vez el campo de baja en 'NULL'.

4. Cuando se acaba un periodo de activación y el usuario se expulsa del sistema, se busca en el histórico una fila con el valor de ese login y el campo de baja igual a 'NULL'. En esa fila hay que poner el campo de baja igual al momento actual.

De esta manera los usuarios van a permanecer en la tabla *activacion*, lo único que se cambiará sera sus periodos de activación. En la tabla *hco_activacion* se guardarán los periodos que ha estado activo con sus parámetros correspondientes.

Histórico de conexiones

En la tabla *hco_conexiones* se almacena un histórico de todas la conexiones que se producen. Esto nos sirve al igual que el histórico de activación para facturar y llevar una estadística. Los campos que se han definido y sus tipos correspondientes en MySQL son:

cod	INT(11) auto_increment
login	VARCHAR(8)
tarifacion	ENUM("pdiaria","pmensual","pospago","bono","prepago")
inicio	DATETIME
fin	DATETIME
id_cola	VARCHAR(4)
ubicacion	VARCHAR(17)
tarifa	FLOAT(4)
facturado	ENUM("si","no")

En este histórico también se insertan los valores cuando comienza la conexión, y cuando la conexión termina se actualiza el valor del campo fin en el histórico y se borra la fila de *conexiones*. De esta manera, al facturar, tenemos la ventaja de que las conexiones activas están en el histórico y podemos decidir si facturar o no esa conexión, esta cuestión se verá en el apartado correspondiente en la sección de administración.

3.1.3. Aplicaciones principales

Para conseguir que las tablas se rellenen correctamente y los usuarios puedan acceder al sistema se han redactado varios scripts en Perl y un módulo ControlNocat.pm que contiene las funciones utilizadas. Hay dos scripts que se ejecutan para registrar las conexiones realizadas. Concretamente los scripts *conexion* y *salida*, que son los encargados de rellenar la tabla de conexiones y el histórico de conexiones.

Además de estos dos scripts, existen otros dos que controlan el acceso de los usuarios al sistema. Estos dos scripts son *cn_min* y *cn_hour*. *cn_min* se encarga de comprobar la tabla de conexiones activas para ver si algún usuario ha llegado a su fin y hay que desconectarlo. *cn_hour* se encarga de comprobar que en la tabla *member* de la base de datos nocat, están los usuarios que están en un periodo activo en ese momento. Es decir, tiene que insertar los usuarios cuando entran en un periodo de activación, y borrarlos cuando salen del mismo. A continuación se explica el funcionamiento de los scripts con más detalle.

- conexion. Este script se invoca en el servidor de autenticación desde la pasarela cada vez que un usuario se conecta. Cuando el usuario ha sido autenticado, y la pasarela ha insertado la nueva regla en el cortafuegos para permitir navegar al usuario, la pasarela invoca por ssh el script conexion, que se ejecuta en el servidor de autenticación. Este script llama a la función conexion del módulo ControlNocat que lee de la tabla *activacion* los datos del usuario, concretamente los campos tarifacion, baja, bono y tarifa. Seguidamente guarda en la tabla *conexiones* los datos correspondientes:

  - El login es un parámetro que se pasa desde la pasarela.
  - Para rellenar inicio utiliza la función NOW() que proporciona MySQL,
  - En fin guarda el campo baja que ha consultado en *activacion* en todos los casos excepto en el caso de que la conexión sea de tipo pospago, en cuyo caso se almacena el valor NULL. Si es de tipo bono, se le suma el campo bono de *activacion* al momento actual y se inserta el resultado en el campo fin.
  - La ubicación es un parámetro que se le debe pasar desde la pasarela.
  - La tarifa se rellena desde el valor que se consulta en *activacion*.
  - El campo facturado se rellena con el valor 'no'. Este campo sólo se utiliza para las conexiones pospago y como es lógico no se puede facturar hasta que la conexión termine.

  En la figura 3.2 se observa el funcionamiento del script conexion[3].

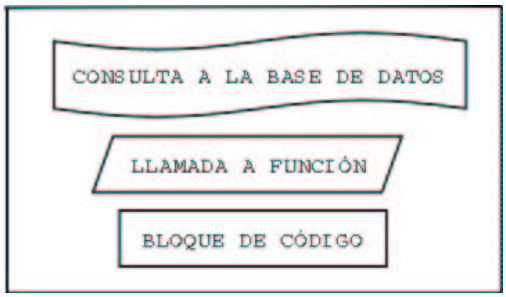


Figura 3.1: Leyenda de los diaframas de flujo

- salida. Este script, al igual que conexion se invoca desde la pasarela, y se invoca justo después de eliminar las reglas del cortafuegos que permitían el acceso de dicho usuario. Esto se puede producir por varios motivos que son: primero que el usuario pulse el botón de logout, en segundo lugar porque el usuario cierre el popup y la conexión expire, o en tercer lugar cuando nosotros borramos del

[3]En la figura 3.1 mostramos la notación usada en el diagrama de flujo

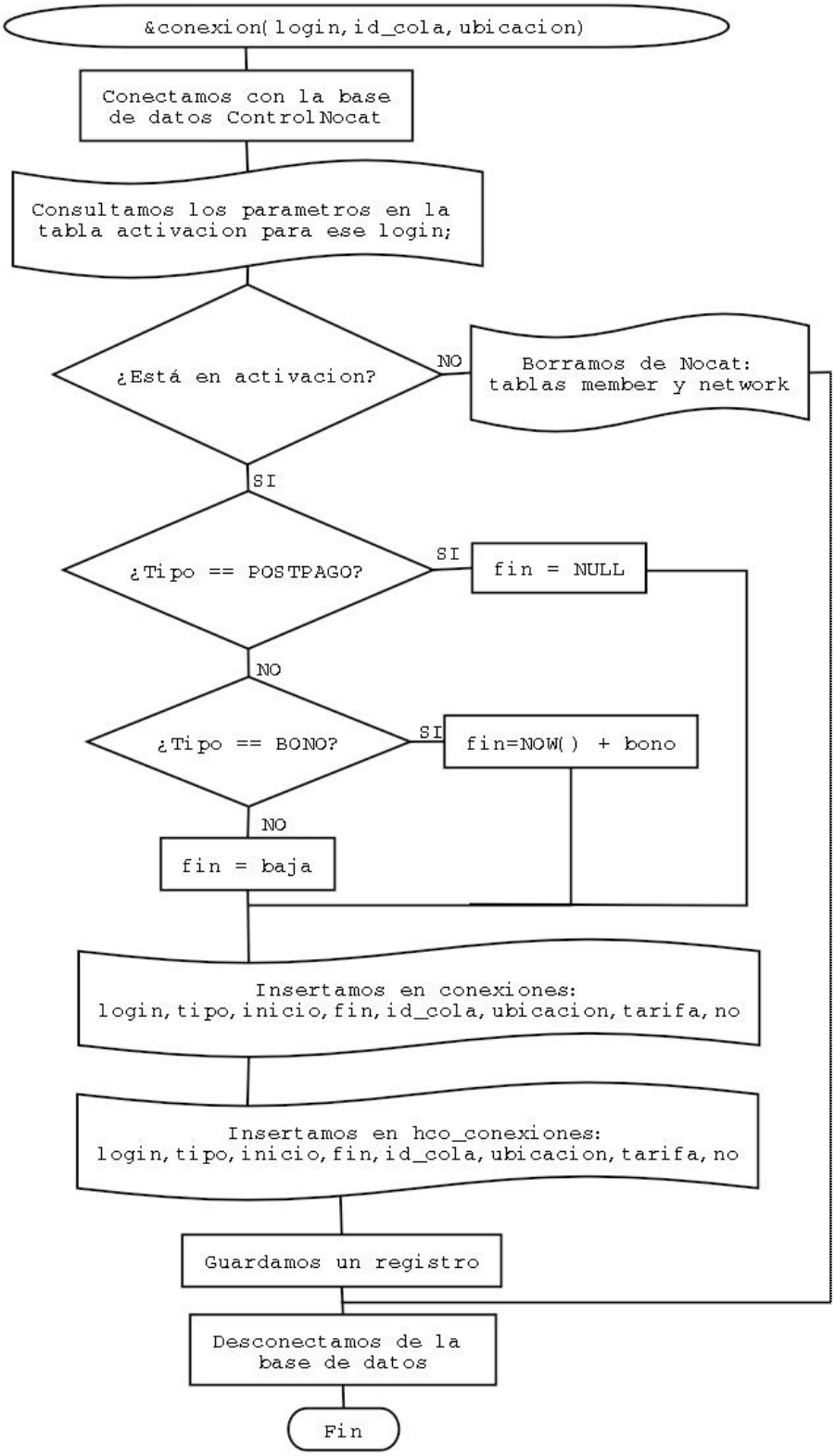


Figura 3.2: Conexion

servidor a dicho usuario y el servidor indica a la pasarela que debe cortarle el servicio. Cuando una de estas tres situaciones ocurre la pasarela invoca el script salida con un parámetro que se corresponde con el login del usuario que acaba de abandonar el sistema.

El script salida llama a la función salida del módulo ControlNocat.pm que borra la fila correspondiente en *conexiones* y copia el valor en el histórico de conexiones. En el campo fin se vuelve a utilizar la función NOW() que nos ofrece MySQL. En el caso de que la conexión fuera de tipo bono, se calcula la duración de la misma y se resta al valor de bono en la tabla *activacion* la duración de la conexión. De esta manera la siguiente vez que se conecte en la tabla *activacion* seguirá estando el tiempo que le queda por consumir. En el caso de que la duración haya sido mayor que el valor que le quedaba por consumir se almacena 00:00:00. Este caso puede ocurrir ya que el sistema que vamos a utilizar no será exacto y puede dejar "un minuto de gracia" a los usuarios, este tema se verá en el siguiente punto. El funcionamiento del script salida está descrito en la figura 3.3.

- cn_min. Es un proceso que se demoniza[4], y se conecta con la base de datos ControlNocat. Lo que hace este proceso es llamar a la función check_1min una vez por minuto, concretamente está ajustado para que lo haga cuando los segundos marcan 00. Véase la figura 3.4.

  La función check_1min[5] comprueba en la tabla *conexiones* si existe alguna conexión en la que el campo fin sea anterior o igual al momento actual, si esto ocurre quiere decir que hay que cortar el servicio de este usuario. Para cortar el servicio se borra al usuario de la tabla *member* y de la tabla *network* de la base de datos nocat. Al borrar al usuario de las tablas de nocat, cuando el popup intente renovar las credenciales del usuario será rechazado y no se enviará nada a la pasarela. De tal manera que cuando expire el tiempo predeterminado en la pasarela el usuario dejará de tener acceso a Internet. Debido a esto, el tiempo que se pondrá en la pasarela será el mínimo permitido que es un minuto, para que el usuario pueda navegar como mucho durante un minuto más de lo contratado.

  Además de quitarlo de la tabla *member* de nocat, hay que registrar en el histórico de activación que ha dejado de estar activo. Lo único que hay que hace es actualizar el valor de baja en el histórico con la función NOW(). Cuando el login es borrado de la tabla *member* se desencadena un proceso que llevará a la pasarela a borrar las reglas del cortafuegos que permitían el acceso de ese usuario, después de borrar esta regla se invocará el script salida que será el encargado de borrar de *conexiones* y pasar al histórico en ese momento.

- cn_hour[6]. Es un script que se conecta con las bases de datos ControlNocat y nocat, y hace las siguientes comprobaciones:

[4]Proceso que se ejecuta en background y no tiene padre.
[5]Véase la figura 3.5.
[6]Véase figura 3.6.

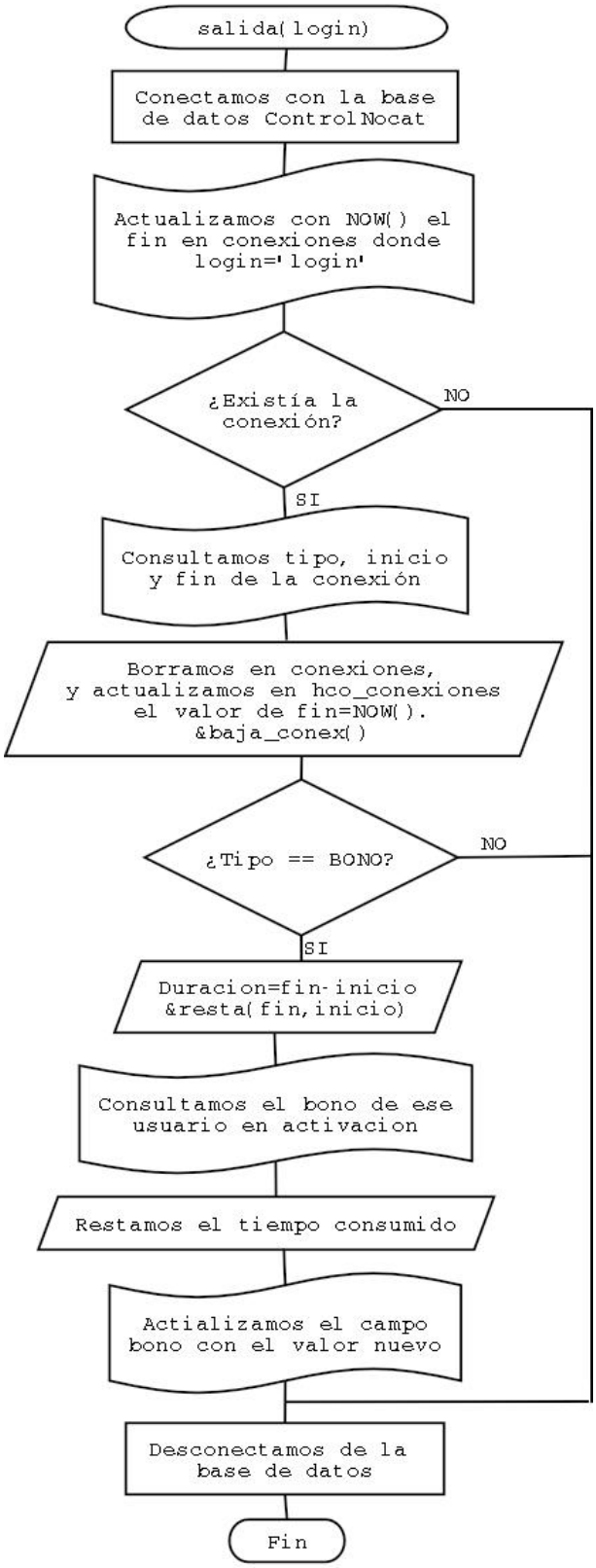


Figura 3.3: Salida

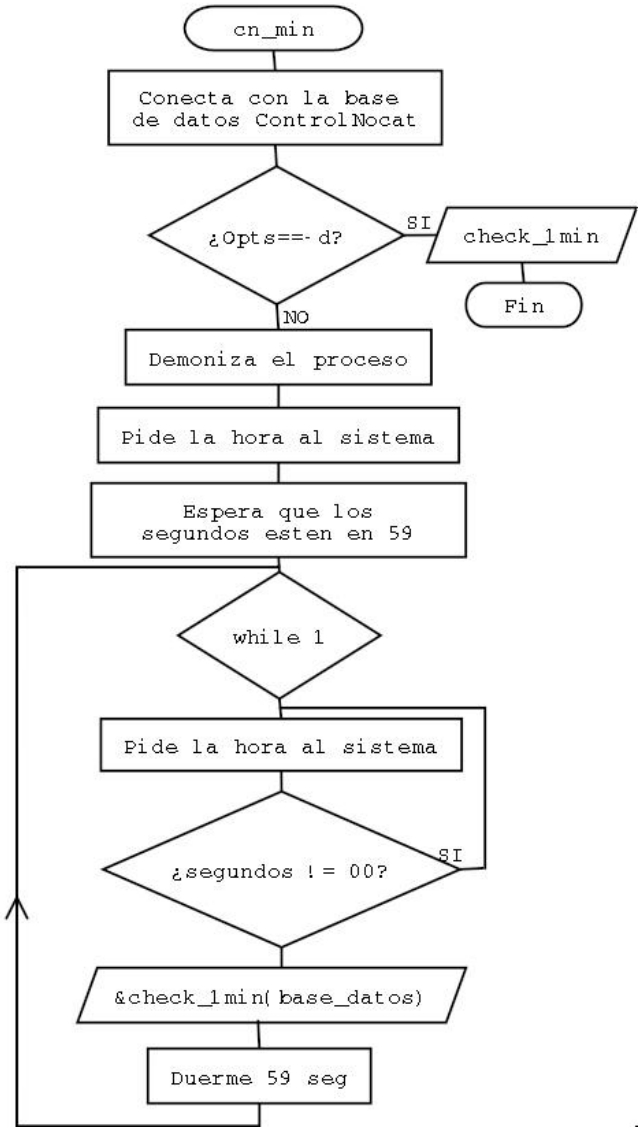


Figura 3.4: cn_min

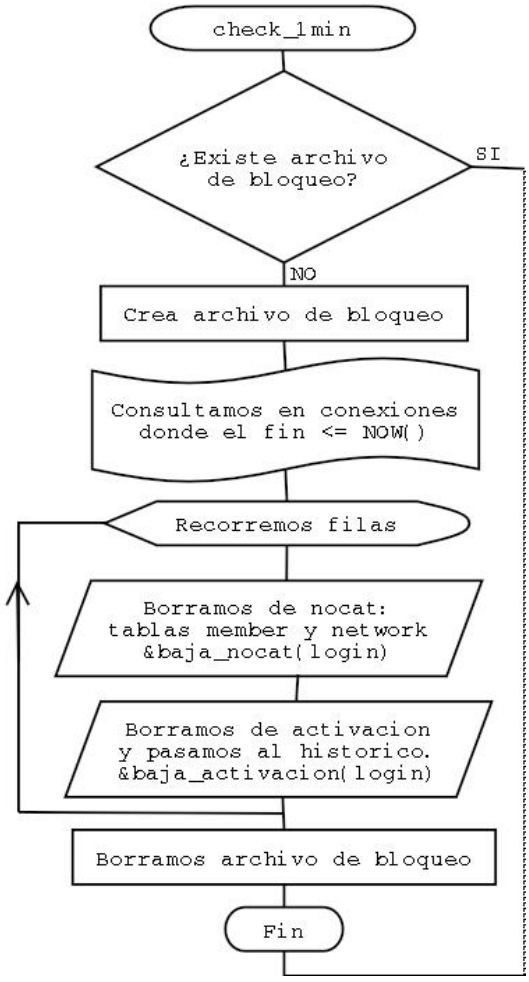


Figura 3.5: check1_min

1. Consulta los usuarios que están dentro de la tabla *member*.
2. Consulta en *activacion* las entradas donde el campo baja es anterior al momento actual, o bien que el campo bono sea igual a 00:00:00. En cualquiera de estos dos casos lo borra de las tablas *member* y *network* de nocat para que no pueda seguir navegando. Asimismo, se actualiza el valor de baja en el histórico. Al borrarlos de la base de datos nocat, si el usuario está conectado, cuando pase un minuto se le denegará el permiso y será invocado el script de salida desde la pasarela. El script de salida será el que se encargue de quitar de la tabla *conexiones* y pasar al histórico.
3. Comprueba si existe algún usuario que este dado de alta en *activacion* y no esté en la tabla *member* de nocat. En ese caso inserta ese login tanto en la tabla *member* como en la tabla *network* de la base de datos nocat.

Este script será ejecutado una vez por hora y para ello programamos una tarea cron[7], de forma que se ejecute una vez cada hora en el minuto que nosotros queramos.

Hemos visto que a cada minuto se comprueba la tabla *conexiones*, mientras que a cada hora se comprueba la tabla *activacion*. La pregunta que nos podemos hacer es: ¿por qué no se comprueban las dos tablas en cn_min una vez cada minuto? La respuesta a esta pregunta es que en la tabla *conexiones* sólo están las conexiones activas en un momento determinado y la tabla será relativamente pequeña, sin embargo en la tabla *activacion* puede haber muchos más usuarios. Debemos tener en cuenta que el servidor de autenticación es centralizado y puede contener la información de muchos usuarios. Imaginemos por ejemplo el caso de una cadena de hoteles con un único servidor para todas sus sedes, en la tabla *activacion* puede haber miles de entradas. Si se consulta esta tabla a cada minuto podríamos congestionar el sistema, debido a esto se ha decidido que a cada minuto sólo se comprueben las conexiones activas cuyo tamaño será menor. Además es imprescindible consultar esta tabla a cada minuto ya que hay que controlar que un usuario no se conecte si no ha pagado. Si un usuario permanece durante una hora en la tabla *activacion* debiendo estar fuera no pasa nada ya que esto no supone que esté navegando, de hecho en el momento que intente navegar se borrará de la tabla *member* y se pasará al histórico. Esto último se realiza en la función check_datos que la veremos en el siguiente punto.

[7]Para más información véase el apéndice A.2.

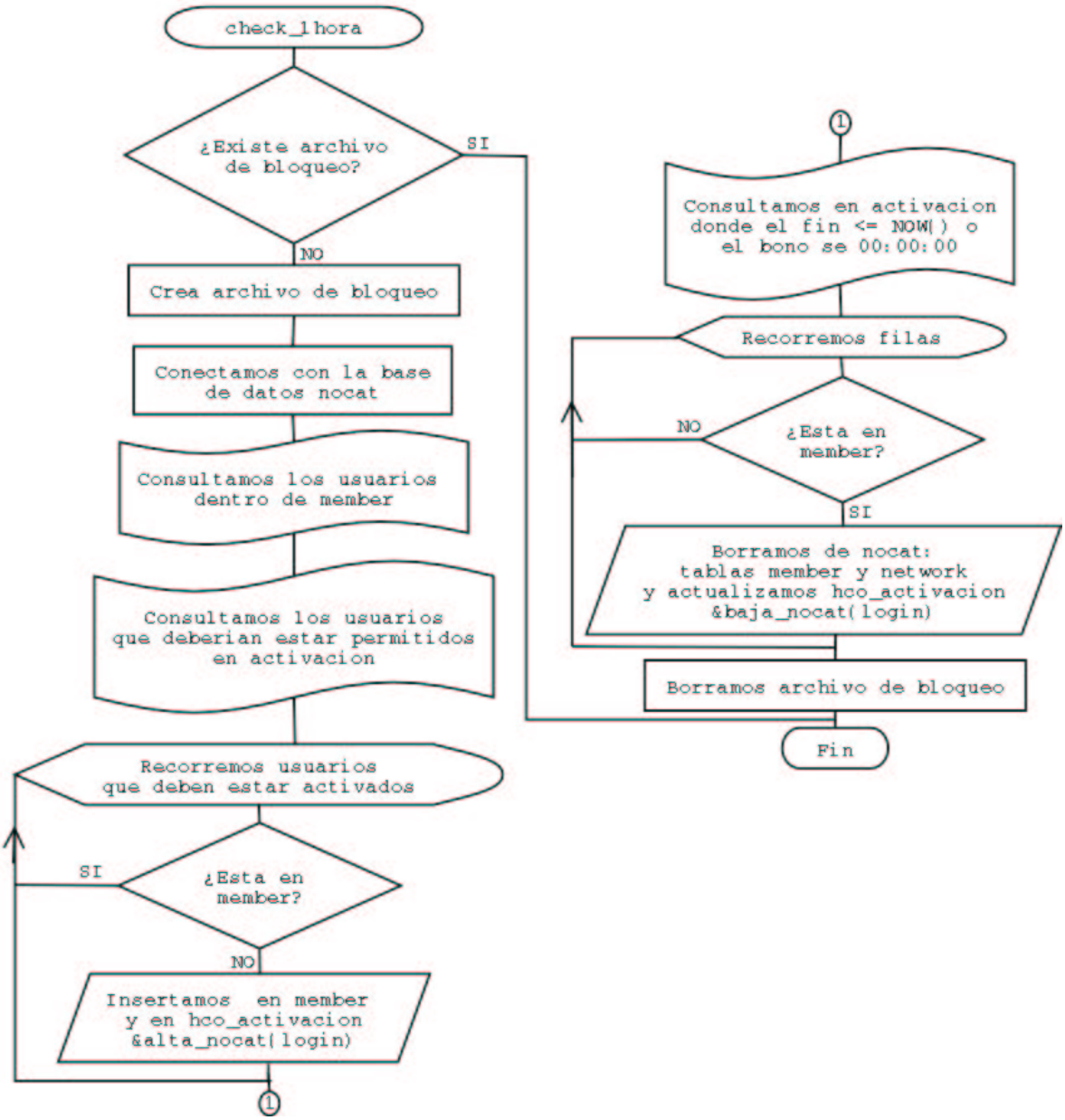


Figura 3.6: check1_hora

3.1.4. Funciones auxiliares

- check_datos[8].El sistema Nocat de autenticación no proporciona control de estado, es decir, no guarda en ningún lugar qué usuarios están conectados. Teniendo esto en cuenta el sistema no es capaz de darse cuenta si todos los clientes se autentican con el mismo login. El servidor de autenticación cada vez que recibe una petición de un login, si está en la tabla *member*, lo acepta sin hacer ninguna comprobación. Nosotros hemos creado la función check_datos para comprobar varios parámetros.

  La función check_datos es llamada desde login.cgi que es el script que se ejecuta cuando un usuario se autentica. Después de que este script compruebe que el usuario existe y guarde a qué grupo pertenece, se llama a la función check_datos con los parámetros login y ubicación.

  La función check_datos comprueba que el usuario ya ha entrado en su periodo de alta, que todavía no ha salido del periodo y también que no haya agotado el bono. Si cumple estas condiciones comprueba que no hay ninguna conexión activa con ese login, para ello consulta la tabla *conexiones*. Esta comprobación se hace para evitar que un usuario comparta su login y puedan entrar varias personas a la vez con el mismo login. Si no existe ninguna conexión, se comprueba que la ubicación desde donde se está intentando conectar coincide con la ubicación que dicho usuario tiene en la tabla *activacion*, o bien, que en la tabla *activacion* el campo ubicacion tiene valor "NULL", y entonces el usuario se puede conectar desde cualquier pasarela.

- resta. Esta función se utiliza en la función salida para restar dos tiempos. Los formatos de entrada pueden ser AAAA-MM-DD HH:MM:SS o también HH:MM:SS, la salida de la función es una cadena con formato HH:MM:SS resultado de restar el primer parámetro menos el segundo. Es utilizado para calcular la duración de una conexión del bono y para calcular el tiempo que le queda por consumir al bono.

- log. Función que recibe un mensaje como parámetro y lo saca en el syslog.

- alta_nocat. Esta función se encarga de insertar un usuario dentro de las tablas *member* y *network*. Además inserta la fila correpondiente a dicho usuario dentro del histórico de activación. Esta función será invocada desde check_1hora y desde alguna aplicación de administración cuando un usuario se da de alta y está en ese momento activo.

- baja_nocat. Función que da de baja en la base de datos nocat. Borra de las tablas *member* y *network* al usuario que se le pasa como parámetro. Esta función es invocada desde check_1min, check_1hora y check_datos. Además dentro de esta función es donde cerramos el histórico de activación.

[8]Véase figura 3.7.

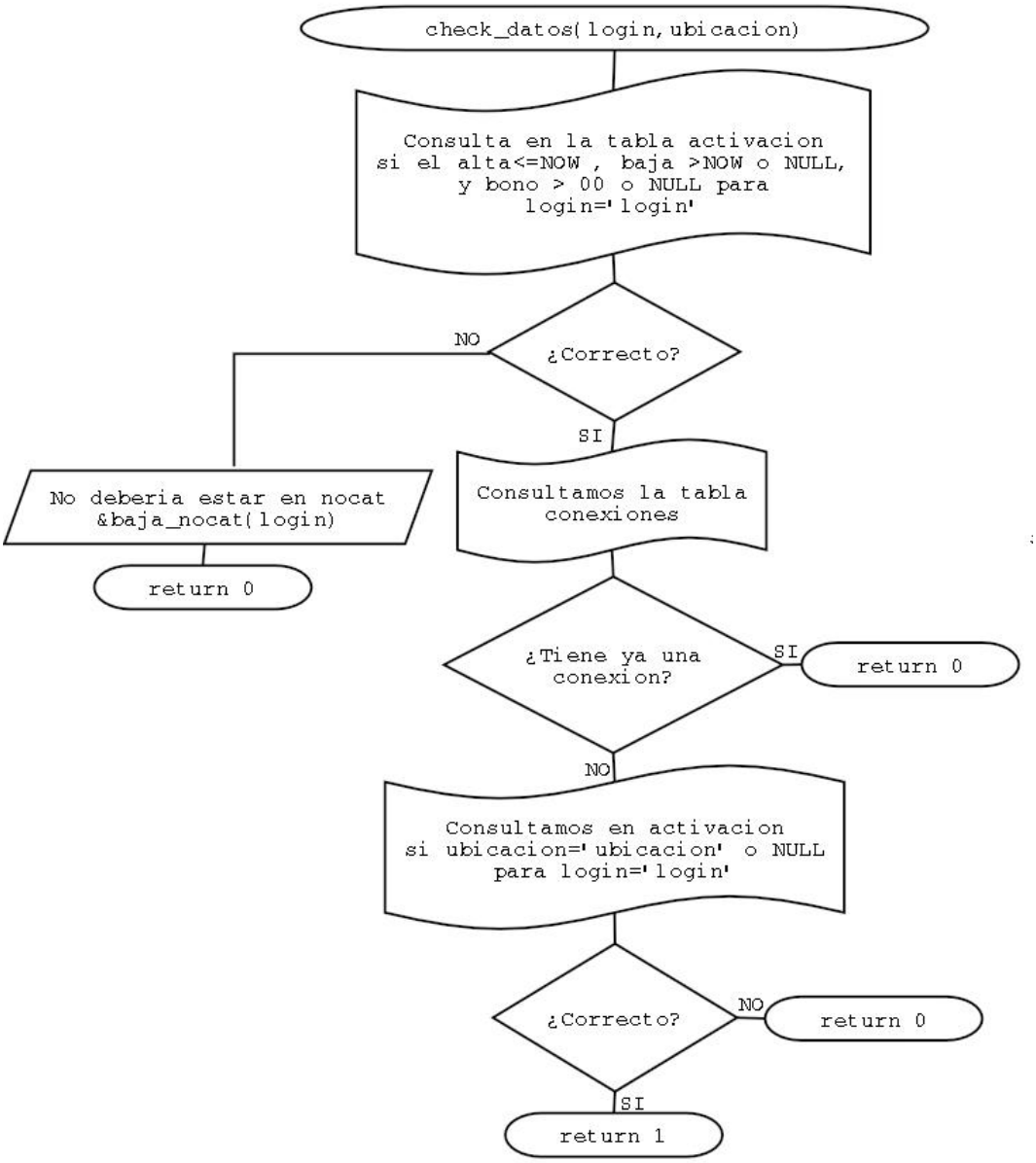


Figura 3.7: check_datos

- baja_conex. Función que borra de la tabla *conexiones* al usuario que se pasa por parámetro y lo inserta en el histórico. Como primer parámetro se le pasa el manejador de la base de datos ControlNocat y como segundo el login a dar de baja. Esta función sólo es invocada desde salida.

3.2. Calidad de servicio

En este apartado se contempla la necesidad de ofrecer una calidad de servicio a los usuarios del sistema. En el primer punto definimos las necesidades del sistema y la situación de partida, y en el segundo punto presentamos la solución adoptada.

3.2.1. Necesidades del sistema

Vamos a intentar dotar al sistema de un método para repartir el ancho de banda del que disponemos.

Nocat incluye tres clases[9] para el reparto del ancho de banda. A estas tres clases se le asignan los usuarios de las clases Public, Member y Owner. Para lograr esto, en el momento de arrancar la pasarela, Nocat llama un script de iniciación que crea tres clases distintas para la interfaz local y otra tres para la interfaz de salida. Estas tres clases definen tres calidades de servicio. Los parámetros que se pueden definir son el caudal de bajada y subida de la clase Owner, caudal de bajada y de subida de la clase Member y caudal de bajada y de subida de la clase Public. Dependiendo de la clase a la que pertenezca un usuario tendrá unos parámetros en su conexión.

Para nuestros intereses el sistema que proporciona Nocat es muy rígido y sólo permite tener tres clases predefinidas, con lo que un usuario no podría tener unos parámetros de conexión personalizados. Nuestro objetivo es poder definir para cada usuario un caudal mínimo garantizado, un caudal máximo alcanzable y una prioridad. De esta manera cada usuario puede contratar un servicio personalizado que, por supuesto, se le aplicará una tarifa conforme al servicio contratado.

3.2.2. Solución adoptada

NoCatAuth utiliza la disciplina de colas cbq (Class Based Queueing), que es la disciplina *más compleja, la más publicada, la menos comprendida, y probablemente la más difícil de configurar correctamente.*[10] En lugar de eso nosotros hemos elegido HTB que funciona igual que CBQ pero es mucho más sencilla de configurar. Esta disciplina de colas divide el enlace real en varios enlaces simulados, y asigna a cada uno de ellos un ancho de banda mínimo, un ancho de banda máximo y una prioridad. La particularidad que presenta esta disciplina de colas es que te permite definir todas las clases que quieras aunque la suma de los anchos de banda mínimos sea mayor que al ancho de banda real del enlace. Cuando esto ocurre el ancho de banda lo reparte

[9] El término clase se define dentro del tema de disciplina de colas como una de las partes en las que dividimos en ancho de banda.

[10] Cita del LARTC Linux Advanced Routing & Traffic Control.

proporcionalmente según los anchos de banda mínimos que tengan definidos las clases. Esto nos asegura que el usuario que contrata más caudal mínimo, tenga más ancho de banda que el que paga menos. Cuando todas las clases definidas están transmitiendo al valor de su tasa garantizada, y todavía sobra capacidad disponible del total del enlace, se reparte este ancho de banda entre todas las clases dependiendo del caudal mínimo que tengan garantizado. En realidad el ancho de banda se reparte entre las clases de mayor prioridad y teniendo en cuenta el valor del parámetro *quantum*. No se va a especificar ningún valor de quantum manualmente, ya que HTB toma por defecto un valor proporcional al caudal mínimo garantizado. De esta manera, los usuarios que tengan un ancho de banda mínimo contratado mayor, tendrán un aumento mayor cuando esté disponible más ancho de banda.

Para más información acerca de la disciplina de colas HTB véase el apéndice A.4.

Esquema utilizado

La esquema que se va a utilizar consiste en crear una nueva clase para cada usuario que entre en la red, de esta manera, los parámetros son completamente ajustables a los usuarios. Cuando el usuario sale del sistema se vulve a borrar la clase. De esta manera tendremos una clase por cada usuario dentro del sistema en la interfaz interna. En la interfaz externa sólo se ha creado una clase, de manera que no hay reparto del ancho de banda de subida.

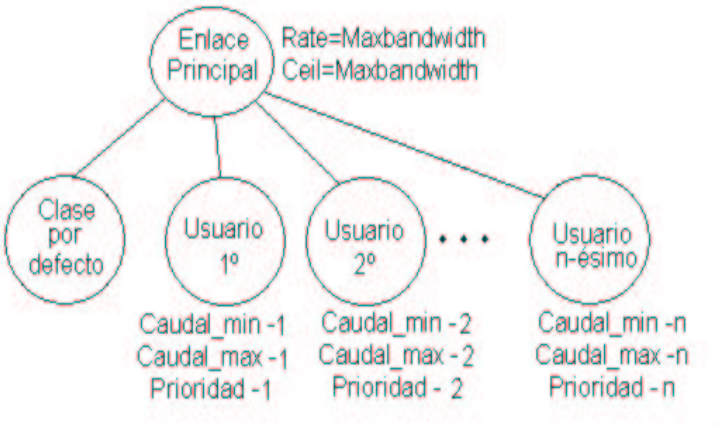


Figura 3.8: Esquema de clases

Funcionamiento

Al iniciar la pasarela se ha sustituido el script que incluía Nocat por un script llamado inicio_QoS que crea las siguientes clases:

- Root Down. Es la clase raíz de la interfaz interna. De esta clase colgarán todas las clases definidas en esta interfaz. El ancho de banda asignado a esta clase es el total del enlace. El script inicio_QoS toma los valores del archivo de configuración de Nocat, donde se ha creado una directiva llamada MaxBandWidhtDown.
- Root Up. Esta es la clase raiz para el caudal de subida, al igual que para la bajada toma los valores del archivo de configuración.
- Default Down. Es la primera clase hijo de Root Down y también toma los valores del archivo de configuración de nocat. Está definida para recoger todo el tráfico que no sea asignado a ninguna clase. A esta clase se le asigna como tasa garantizada MinBandWidthDown y como máximo alcanzable el total MaxBandWidthDown.
- Default Up. Es la clase por defecto para la subida. En principio es la única que se define y por tanto todo el tráfico de subida pertenecerá a esta clase. Los valores serán MinBandWidthUp para la tasa garantizada y MaxBandWidthUp para el máximo alcanzable.

Una vez definidas estas clases inicialmente, sólo nos queda que cada vez que un usuario se conecta se cree una nueva clase con sus parámetros correspondientes. Para crear y borrar las clases se ha creado un script qos.fw al que hay que pasarle los siguientes parámetros: un identificador de clase, la dirección IP del usuario, el ancho de banda mínimo, el ancho de banda máximo y la prioridad. Todos los parámetros son obligatorios en el caso de que estemos creando una clase y sólo el identificador de clase para borrarla. Este script se encarga de crear la clase correspondiente de un usuario, de asignarle una disciplina de colas HTB con los parámetros pasados y de asignar a esa clase todo el tráfico cuyo destino sea la dirección IP del usuario. Al borrar sigue los pasos inversos: borra el filtro para asignar el tráfico, luego borra la disciplina de colas y por último borra la clase.

Ya hemos visto lo que hace el script qos.fw, pero no sabemos cómo pasarle los parámetros. Para pasarle los parámetros se ha usado el mensaje encriptado que devuelve el servidor de autenticación a la pasarela. Dentro de este mensaje introducimos los parámetros: id_cola, rate, ceil y prio que son identificador de clase, ancho de banda mínimo, ancho de banda máximo y prioridad respectivamente. Para asignar el id_cola de una conexión se pensó en primer lugar en utilizar un campo de la tabla *activacion* donde el administrador insertara el identificador con un valor entre 3 y 65535[11], pero

[11] El 1 no es válido y el 2 está asignado a la clase por defecto.

esta solución restringía el número de usuarios activados para un servidor a 65532. Este valor, a priori, no sabemos si será suficiente o no, pero en cualquier caso se puede usar otro método mucho más eficiente. La otra opción, que finalmente ha sido la adoptada, es asignar los identificadores dinámicamente. Para ello se ha creado una "pila" de la que cogemos un identificador cuando un usuario se conecta, y lo volvemos a meter en la pila cuando un usuario se desconecta. De esta manera, la limitación impuesta al sistema es mucho menor, ya que limitamos a que haya 65532 usuarios conectados a la vez que dependan del mismo servidor de autenticación. Para guardar el id_cola de cada conexión y así poder borrar la clase cuando se desconecte utilizamos el campo de la tabla *conexion* id_cola. Como pila se usa un nueva tabla que incluimos en ControlNocat llamada id_cola que tiene un solo campo y es iniciada con los valores de 3 a 65535 en hexadecimal. Para extraer de la pila seleccionamos un valor de una fila y después la borramos, y para volver a meter en la pila introducimos otra vez el valor en la tabla.

Los parámetros de caudal y prioridad están guardados en la tabla *grupos* en los campos caudal_min, caudal_max y prioridad. La tabla *grupos* se usa para definir todos los grupos que habrá en el sistema. Los campos que incluye la tabla *grupos* son los siguientes:

cod_grupo	VARCHAR(20)
tarifacion	ENUM("pdiaria","pmensual","pospago","bono","prepago")
tarifa	FLOAT(4)
inicio	DATETIME
fin	DATETIME
bono	TIME
caudal_min	INT
caudal_max	INT
cod_politica	VARCHAR(3)
prioridad	VARCHAR(1)
comentarios	VARCHAR(250)

Para meter los parámetros en el mensaje encriptado, dentro del script login.cgi[12] que es el que autentica al usuario cada minuto, se inserta una función llamada consultaQoS que se incluye en el módulo ControlNocat.pm. Esta función consulta los parámetros caudal_min, caudal_max y prioridad en la tabla *grupos* y consulta bien el id_cola de la tabla *conexiones* si el usuario estaba activo, o bien saca un nuevo id_cola de la tabla de id_cola que será asignado a ese usuario. De esta forma a cada minuto se envía un mensaje encriptado con los valores del servidor de autenticación, esto nos sirve para poder cambiar los datos dinámicamente durante una conexión.

[12]Para más detalles véase el apéndice A.1.

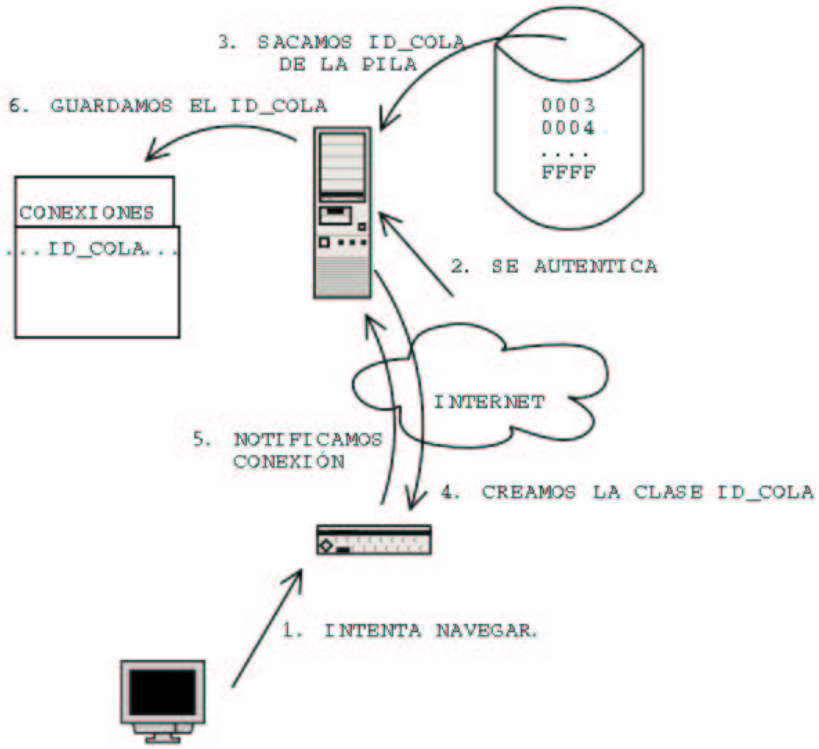


Figura 3.9: Proceso de conexión

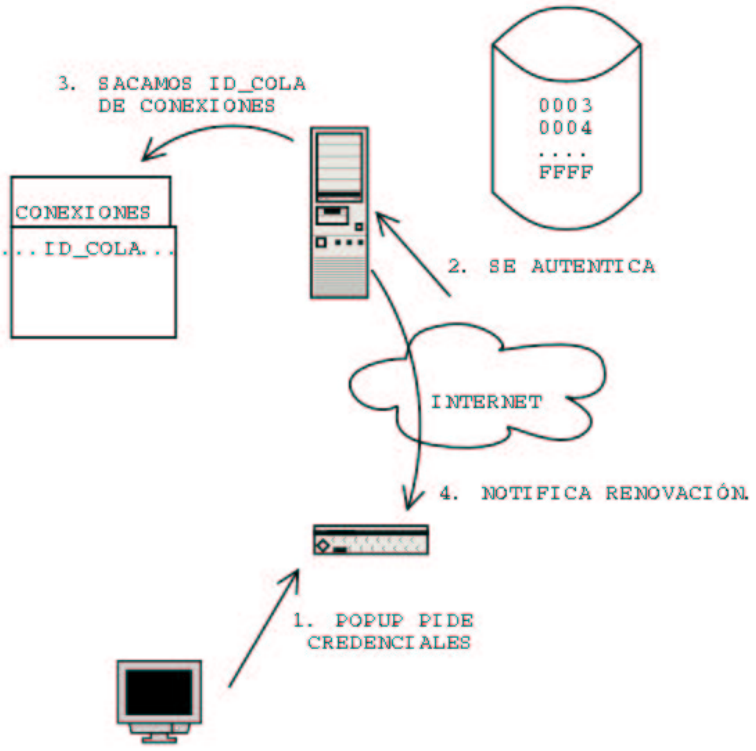


Figura 3.10: Proceso de renovación del popup

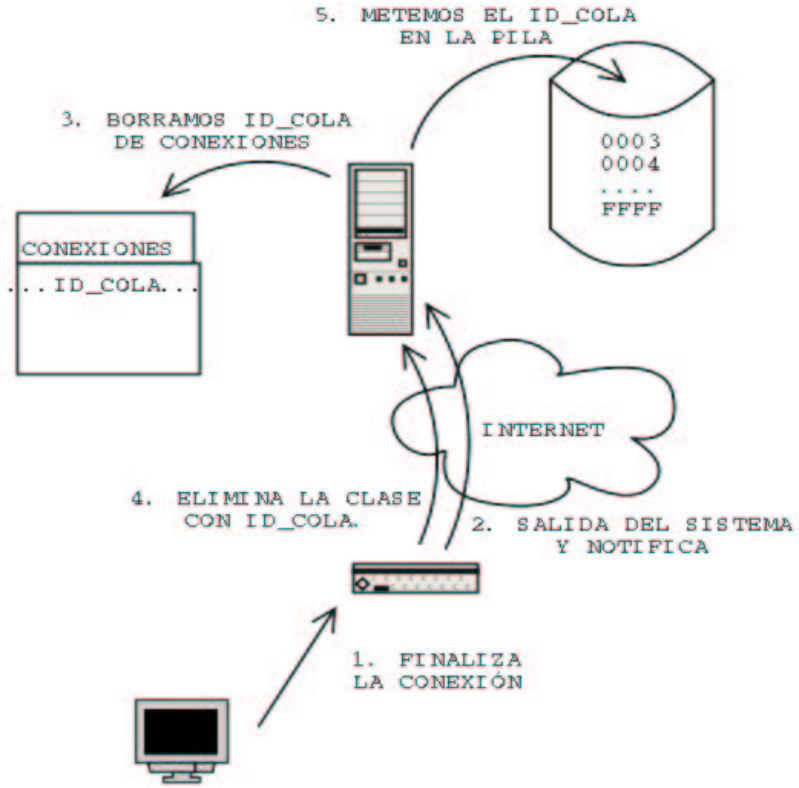


Figura 3.11: Proceso de salida del sistema

Cambios en los parámetros de conexión

Como hemos dicho antes los parámetros de calidad de servicio van incluidos en el mensaje encriptado que se envía desde el servidor hasta la pasarela. Aprovechando esta circunstancia, a cada minuto se reciben en la pasarela los nuevos prámetros de calidad de servicio, hemos cambiado el método permit[13] de la clase gateway en la pasarela para comprobar si estos parámetros cambian con respecto a los anteriores. En tal caso se borra la clase anterior y se crea una nueva con los nuevos parámetros. Para ver los detalles de implementación vea la sección 4.2.

3.2.3. Limitaciones

Este sistema tiene dos limitaciones:

- El primer problema que presenta esta técnica es que el caudal mínimo garantizado deja de estar garantizado si el administrador del sistema no da valores adecuados a este parámetro. Para garantizar que un usuario va a disponer de ese caudal mínimo el administrador debe asegurarse que la suma de los caudales mínimos de todos los usuarios que soporta el sistema, no supere el valor del ancho de banda disponible total del enlace. No obstante el sistema sigue funcionando aunque el administrador asigne mal estos valores. La diferencia es que este parámetro deja de tener el significado de tasa garantizada, y pasa a ser un valor relativo para definir la proporción con la que se reparte el ancho de banda entre las clases.

- Por otro lado, nos hemos centrado únicamente en uno de los dos enlaces que atraviesa la comunicación. Los enlaces a tener en cuenta son: en primer lugar desde los PCs hasta la pasarela, y en segundo desde la pasarela hacia Internet. Esta técnica hace que se reparta el ancho de banda entre los usuarios en el sistema en el enlace con Internet, pero no dentro de la red inalámbrica.

  En circunstancias normales la velocidad de transmisión dentro de la red local no debe suponer ninguna limitación sobre la comunicación, ya que en la 802.11b a 11Mbps se logran velocidades de hasta 1Mbps. No obstante, esto depende de muchas circunstancias, tales como el número de usuarios, la distancia al punto de acceso, las condiciones atmosféricas, etc... Por este motivo no se pueden ofrecer garantías acerca del ancho de banda, pero al menos si se puede garantizar que el que pague más recibirá mejor servicio.

Una vez expuestas estas limitaciones, se puede buscar una solución específica dependiendo del tipo de red local. Habría que estudiar la red que se está utilizando, por

[13]Véase apéndice A.1 para ver los detalles del funcionamiento de Nocat

ejemplo, puede ser la norma 802.11.a a 54Mbps, o puede ser que el sistema esté instalado dentro de una red cableada Ethernet 10/100, o la 802.11b a 11Mbps o a 1Mbps. También habría que contemplar el número de usuarios que va a soportar la red. En función de estos parámetros se puede establecer un valor mínimo para el caudal mínimo que sí se podría garantizar.

3.3. Políticas de acceso

En este apartado se estudia la manera de controlar los servicios a los que se permite acceder a los usuarios.

3.3.1. Necesidades del sistema y estado previo

La característica que incluye el sistema Nocat referido a este tema es muy limitada y consiste en tener dos directivas definidas en el archivo de configuración: IncludePorts y ExcludePorts. Son excluyentes, es decir, si tienes definido IncludePorts se ignora ExcludePorts, y su funcionamiento es el siguiente: en Includeports se definen los puertos a los que se le va a permitir acceder a los usuarios de la clase Public, al resto el de puertos el acceso está prohibido; si por el contrario está definido ExcludePorts los miembros de la clase Public podrán acceder a todos los puertos excepto a los que se indique en ExcludePorts.

Este sistema está muy limitado y únicamente sirve para los miembros de la clase Public, nosotros pretendemos crear diferentes políticas para diferentes grupos y tener la capacidad de que cada una defina unos servicios. Por ejemplo habrá una política *Sólo Web* en la que sus miembros sólo podrán acceder al puerto 80 y al puerto 443. Al resto de puertos tendrán el acceso cortado. También se puede definir otras políticas que además del tráfico web permitan el uso de correo electrónico, chat, ftp, etc...

3.3.2. Solución adoptada

Definición de nuevas políticas

Para permitir esta flexibilización del sistema se ha retocado el funcionamiento de Nocat consiguiendo una funcionalidad mucho mayor. Para conseguir esto se han ampliado las cuatro marcas que usaba el nocat: 0x1, 0x2, 0x3, 0x4 y ahora se pueden usar hasta seis marcas más, y cada una de las marcas añadidas corresponde con una nueva política de acceso. Recordando como funcionaba la pasarela de Nocat, teníamos que las reglas del cortafuegos usaban el marcado de paquetes para discriminar los usuarios. Tenían cuatro clases: Owner, Member, Public y no autenticado. Cuando un usuario nuevo entra en el sistema se marcan sus paquetes con la marca del grupo al que pertenece.

Lo primero que hemos hecho ha sido definir en el archivo de configuración seis pares de directivas nuevas. Cada par está formado por: MarkPolicyX y PolicyX, donde la X es un número que va desde 1 hasta 6. MarkPolicyX define la marca de la política X, mientras que PolicyX define una lista con los puertos permitidos para esa política. Una

vez definidas las políticas con sus marcas y sus puertos, es en el momento de iniciar la pasarela cuando se crean las reglas que determinan las políticas. Se ha modificado el script initialize.fw para que en el caso de que se hayan definido las directivas PolicyX, se creen reglas que permitan el acceso a los puertos indicados en PolicyX para los paquetes marcados con MarkPolicyX, y se deniegue el acceso al resto de puertos. El acceso al puerto que utiliza el demonio de la pasarela también se incluye siempre.

Asignación de políticas a los usuarios

Ya están las políticas definidas y las reglas creadas, lo que nos falta es saber qué política tiene contratada un usuario y como marcar sus paquetes con la marca correspondiente.

Para realizar esta tarea hemos aprovechado la funcionalidad de Nocat cambiando una pequeña parte de su código. El sistema Nocat envía en el mensaje encriptado un campo llamado member donde guarda el grupo al que pertenece el usuario. Ese grupo que se manda se saca de la tabla *network* de la base de datos nocat. En la pasarela comprueba si ese campo contiene un grupo y si es así lo mete la clase Member, si no contiene nada, es decir, no pertenece a ningún grupo, lo mete en la clase Public y si pertenece a un grupo propietario lo mete en la clase Owner. Los grupos propietarios se definen en nocat.conf mediante una directiva.

Para aprovechar este mecanismo lo que se hace es guardar en el grupo la MarkPolicyX correspondiente al usuario en cuestión, y se envia a la pasarela la marca que tiene ese usuario en el campo member. En la pasarela, cuando asigna a los usuarios la clase member si pertenecen a algún grupo, lo quitamos, y en su lugar la clase es igual a la marca que recibimos en el mensaje.

De esta manera hemos conseguido saber en la pasarela la marca que corresponde a cada usuario, ya sólo nos falta que al crear la reglas del cortafuegos de ese usuario, le indiquemos que tiene que marcar los paquetes con esa marca. Esto se hace en el script access.fw. Este script recibe cuatro parámetros:

access.fw [permit—deny] [MAC] [IP] [Class]

En Class se le pasa la clase mencionada anteriormente. Antes, los valores que tomaba eran: Owner, Member y Public. Ahora en lugar de Member hemos incluido las clases: MarkPolicyX, donde la X va desde 1 hasta 6. Cuando indicamos una de estas clases los paquetes se marcan con la marca en cuestión.

Los detalles de implementación están en la sección 4.3.

3.4. Administración del sistema

Esta es la sección más mecánica del proyecto pero una de las más importantes. Este trabajo es el que permite que todo lo anterior sea manejable por una tercera persona y proporciona una base para poder comercializar el producto.

En principio se ha elegido desarrollar esta aplicación sobre un entorno web. Esta elección se ha producido por varios motivos:

- Facilidad. Facilidad de implementación y de instalación.
- Rapidez. Por la escasez de tiempo del proyecto, ya que tenía un plazo de entrega muy corto.
- Adecuado para la comercialización. El entorno web es muy adecuado para demostraciones del producto.

La elección del lenguaje de programación para el desarrollo de las páginas fue una difícil disyuntiva. En principio había tres posibilidades: usar la API C, usar la API DBI o usar la API PHP. Los puntos tenidos en cuenta para nuestra elección han sido:

- Entorno de ejecución. Nuestro entorno de ejecución es una interfaz Web, por lo tanto, se puede descartar C, ya que está más orientado a procesos independientes, mientras que Perl y PHP se adaptan mejor a las aplicaciones Web. Además Perl o PHP tienen herramientas mucho más potentes para el procesado de texto, y no hay que preocuparse de la administración de memoria. Perl proporciona el módulo CGI.pm para realizar páginas de forma dinámica, y se puede usar la interfaz DBI para interactuar con MySQL.

  Por otro lado PHP está diseñado para escribir aplicaciones web, así que este es el entorno donde mejor se encuentra. Más aún, el acceso a la base de datos es uno de los mayores potenciales de PHP.

- Rendimiento. El rendimiento para nuestra aplicación no es demasiado importante, ya que nuestros programas para la administración serán ejecutados por un sólo usuario simultáneamente. No obstante no se debe sobrecargar el servidor ya que estará recibiendo constantemente peticiones de autenticación. Se puede decir que en la primera parte del proyecto era mucho más importante el rendimiento, ya que se trataba de programas que se ejecutaban una vez cada minuto, y peticiones de muchos usuarios que había que atender.

  De todas formas nos interesa que el programa funcione rápidamente y el usuario no tenga que esperar que se carguen las páginas una y otra vez.

  Un programa en C, al ser un lenguaje compilado se ejecuta más rápidamente que los programas en Perl o PHP que son lenguajes interpretados. Por otro lado

debemos destacar que el rendimiento de lenguajes interpretados en un contexto Web mejora cuando el intérprete es invocado como un módulo que es parte del mismo servidor web. En nuestra elección se tendrá esto en cuenta.

- Tiempo de desarrollo. El tiempo de desarrollo juega un papel importante en la elección del lenguaje, ya que se dispone de poco tiempo para desarrollar este proyecto. Por este motivo C queda descartado otra vez, ya que se suele tardar más en escribir un programa compilado que uno interpretado. Dos factores contribuyen a esto. Primero, los lenguajes de guión tienden a proporcionar construcciones de nivel más alto. Esto nos permite pensar en un nivel de abstracción más alto, es decir, en lo que queremos más que en los detalles concernientes a cómo hacerlo.

  En segundo lugar, el ciclo de desarrollo tiene menos pasos para lenguajes de guión. Con C, se recorre el ciclo de compilación-edición habitual según se desarrolla la aplicación. Cada vez que se modifica el programa debemos recompilarlo. Por el contrario, con Perl y PHP el ciclo de desarrollo es simplemente edición-prueba. Además el compilador fuerza a más restricciones en el programa y sigue un proceso de verificación más estricto.

  Una de las ventajas adicionales de Perl es la cantidad de módulos que hay ya desarrollados que se pueden usar para cualquier funcionalidad. Además la principal ventaja de Perl en este sentido es que ya se ha estudiado para realizar la primera parte del proyecto. Esto supone que no se necesita tiempo extra en documentarse para aprender PHP.

- Portabilidad. La portabilidad es otro punto muy importante en esta proyecto, ya que en un futuro puede cambiar el motor de base de datos. El programa está pensado para que funcione sea cual sea el motor utilizado.

  En este sentido Perl es el que proporciona una mayor portabilidad, ya que la API DBI de Perl se trata de una interfaz de base de datos genérica. DBI ofrece una abstracción por encima del motor de base de datos que se esté utilizando. Por debajo de la DBI se encuentra el nivel de DBD, que es el controlador de la base de datos propiamente dicho. A este nivel se proporciona soporte para varios motores de base de datos.

Teniendo en cuenta estos puntos hemos optado por desarrollar los guiones en Perl, ya que las ventajas que proporciona PHP no se han considerado superiores a las ventajas que ofrece Perl.

A continuación se irán viendo las diferentes secciones que se han implementado y su funcionalidad.

3.4.1. Tablas de ControlNocat

Para la administración del sistema se han añadido varias tablas en la base de datos ControlNocat. Estas tablas son independientes del funcionamiento del sistema, únicamente tienen una funcionalidad administrativa.

Ubicaciones

En la tabla *ubicacion* se guardan las direcciones IEEE de 48 bits[14] de las pasarelas que hay disponibles desde un servidor de autenticación. La dirección IEEE de 48 bits se guarda en el campo cod_ubicacion, mientras que en descripción se pone el nombre de la pasarela o una etiqueta identificativa que aparecerá en el formulario para activar al usuario.

cod_ubicacion	VARCHAR(17)
descripcion	VARCHAR(15)

El código de ubicación debe contener el mismo valor que se especifica en el archivo de configuración de la pasarela para la directiva ubicación.

Políticas

En la tabla *politicas* se guardan las políticas definidas en el sistema. En el campo cod_politica se guarda la marca que se pasará a la pasarela para marcar los paquetes. En el campo politica se almacena el nombre de dicha política que será la etiqueta que se muestre en el formulario para activar al usuario. Finalmente el campo comentarios se usa para una definición de la política en cuestión.

cod_politica	VARCHAR(3)
politica	VARCHAR(20)
comentarios	VARCHAR(250)

Login

La tabla *login* almacena los pares login/clave. La clave se guarda codificada con el mismo formato que usa Nocat MD5. Esta tabla sirve para almacenar todos los login

[14] Comúnmente se denominan MAC.

del sistema y comprobar si existe un cierto login al dar de alta al usuario. Además es, junto con la tabla *member* de la base de datos nocat, el único lugar donde se guarda la clave.

login	VARCHAR(8)	PRIMARY KEY
passwd	VARCHAR(22)	

Tipos de tarificación

La tabla *tarifacion* se usa para guardar los tipos de tarifación que existen.

cod_tarifacion	ENUM("pdiaria","pmensual","pospago","bono")
descripcion	VARCHAR(250)

Usuarios

En la tabla *usuarios* se guardan los datos personales de un usuario. El campo habitación está incluido para el caso especial de que el producto se instale en un hotel.

login	VARCHAR(8)	PRIMARY KEY
grupo	VARCHAR(20)	
tarifacion	ENUM("pdiaria","pmensual","pospago","bono")	
nombre	VARCHAR(20)	
apellidos	VARCHAR(30)	
dni	VARCHAR(10)	
telefono	VARCHAR(15)	
provincia	VARCHAR(15)	
habitacion	VARCHAR(4)	
comentarios	VARCHAR (250)	

El esquema final de la base de datos con todas sus tablas y referencias está en la figura 3.12. Las claves externas que están en las tablas son claves conceptuales y están implementadas por las aplicaciones que trabajan con las tablas. Es decir, MySQL no soporta las claves externas, por tanto la integridad referencial debemos lograrla con una buena programación de nuestras aplicaciones.

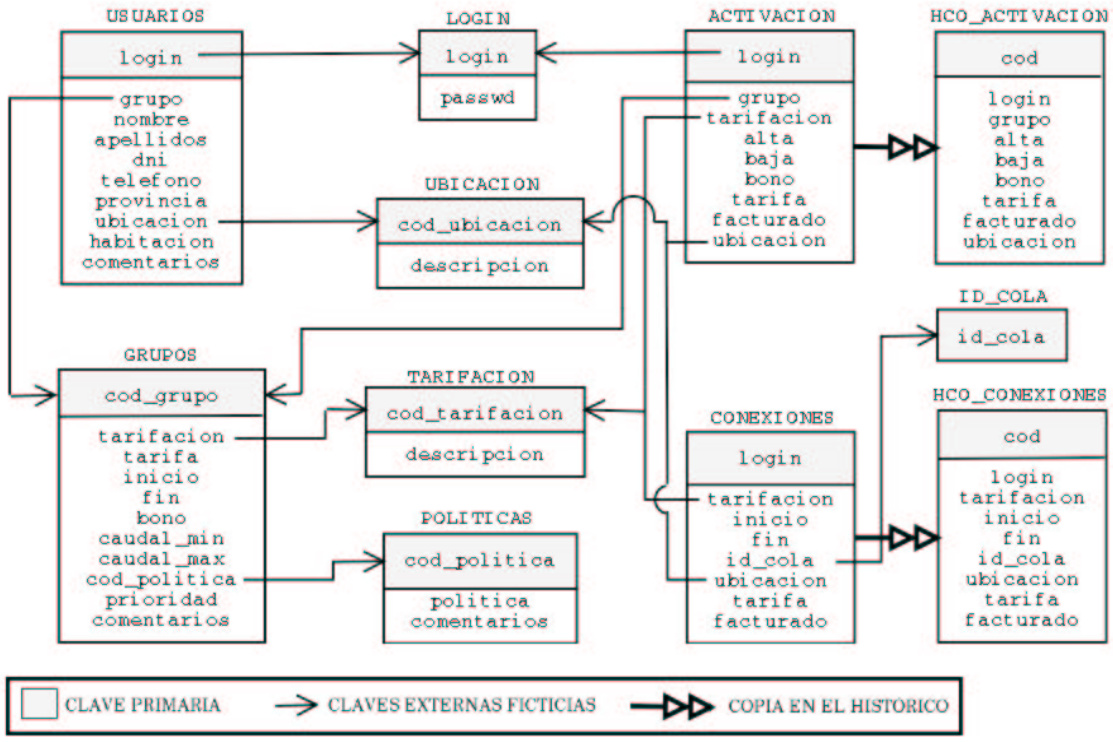


Figura 3.12: Base de datos ControlNocat

3.4.2. Altas

Existen dos tipos de alta: alta de usuario y alta de grupo. Para ambas se han diseñado dos formularios que rellenan las tablas correspondientes.

Alta de grupo

El alta de grupo es lo primero que hay que definir, ya que para dar de alta nuevos usuarios deben pertenecer a un grupo. Para dar de alta a los grupos de ha creado un formulario con el aspecto que se muestra en la figura 3.13.

Figura 3.13: Alta de grupos

Los parámetros caudal máximo, caudal mínimo, tarifa y el nombre del grupo son obligatorios para todos los tipos de tarifación, ya que será en la tabla *grupos* donde se miren los parámetros de la conexión del usuario. Los campos de política y prioridad también son obligatorios, pero como se trata de menús desplegables llevan una opción por defecto. El resto de campos dependen del tipo de tarifación que elijas:

- Plana. Los campos que aparecen son Inicio y Fin. Se tiene la opción de dejarlos en blanco, en cuyo caso cuando se dé de alta a un usuario de ese grupo, se preguntará por unos valores para el inicio y el fin. Si alguno de los campos se rellena para el grupo, no se tiene la opción de cambiarlo cuando se da de alta al usuario.

- Pospago. El campo que se rellena es Inicio. Se puede dejar en blanco, o rellenar con una fecha. La situación es la misma que antes, si se rellena todos los usuarios de ese grupo deben tener esa fecha de alta, si no, para cada usuario de ese grupo se deberá introducir una fecha de alta al activarlo.

- Bono. Para los grupos de tipo bono se introducirá un bono y una tarifa por defecto para el grupo, pero se tiene la posibilidad de cambiar para cada usuario del grupo individualmente.

Para crear este formulario se utiliza un script llamado *alta_grupo*. Se utiliza javascript para esconder los campos que no se usan dependiendo del tipo de tarifación que esté seleccionado.

Alta de usuario

Al crear un nuevo usuario lo primero que se presenta es un formulario para crear un nuevo login. El formulario te da la oportunidad de introducirlo tú mismo o de generar un login aleatorio que no se esté utilizando. El formulario que se ha creado se muestra en la figura 3.14.

Una vez se ha introducido un login que sea correcto y no esté utilizado, se presenta un formulario para rellenar los datos personales del usuario. De todos los datos cabe destacar el grupo. Es muy importante elegir el grupo adecuado ya que los parámetros de conexión del usuario serán los del grupo. El formulario para los datos personales se muestra en la figura 3.15.

Después de introducir los datos personales, nos redirige al formulario de activación. En este formulario nos muestra los parámetros de conexión del grupo y nos permite introducir los datos que no hayan sido prefijados en el grupo. Además incluye el campo ubicación. En el caso de que se introduzca un valor distinto a "Cualquiera" el usuario sólo se podrá conectar desde la pasarela que esté en esa ubicación. El formulario de activación está en la figura 3.16.

Para presentar estos formularios se utilizan dos cgi distintos: *alta_usuario* y *activacion*.

Figura 3.14: Alta de usuarios

3.4.3. Bajas

Al igual que en los procesos de alta se diferencian dos casos: baja de grupo y baja de usuario.

Baja de usuario

Para la baja de usuario se presenta una lista de usuarios y un campo para rellenar. El usuario se puede dar de baja bien pinchando sobre él, o bien escribiendo el nombre en el campo inferior. De cualquiera de las formas se preguntará confirmación, y en caso afirmativo el usuario es dado de baja inmediatamente. Este proceso hace que se expulse del sistema al cliente si estaba conectado, se borre de *activacion* y se borre de la tabla *usuarios*. El único sitio donde queda registrado es en los históricos.

Para realizar esta tarea se ha creado el script *baja_user*. La página generada utiliza javascript para confirmar el borrado. Tenemos un ejemplo de la página generada en la figura 3.17.

Baja de grupo

Para dar de baja a un grupo se presenta una lista con los grupos existentes, y se da la posibilidad de borrarlos pinchando con el ratón encima o introduciéndolos en un campo de texto. Al igual que para los usuarios te pide confirmación.

Figura 3.15: Datos personales

Figura 3.16: Activación del usuario

Figura 3.17: Bajas de usuarios

Cuando se da de baja un grupo se eliminan todos los usuarios que pertenecen a ese grupo.

Para esta sección se ha utilizado el script *baja_group* que tiene un funcionamiento similar al *baja_user*.

3.4.4. Edición

La edición es una de las partes más complejas del sistema de administración, ya que un cambio de grupo de un usuario, o un cambio en los parámetros de un grupo, provoca que haya que cambiar en la tabla *activacion* los datos y registrar los cambios en el histórico.

Edición de usuarios

Para editar un usuario se puede hacer de dos formas: en la sección de edición de usuario o desde una consulta. En la sección de edición se muestra una lista con los usuarios del sistema paginada de diez en diez, también se da la posibilidad de insertar el login directamente. La otra forma es realizar una consulta y desde la lista pinchar en el login del usuario que quieras editar.

Una vez has elegido un usuario para editar se muestra en pantalla el mismo for-

mulario que para dar de alta relleno con los datos del usuario. Además aparecen dos nuevos botones: 'Borrar registro' y 'Cambiar clave'.

Una vez mostrado el cambio en los datos se pasa a los datos de activación. En el caso de que el usuario haya cambiado de grupo aparece en el formulario de activación los parámetros del nuevo grupo.

Para la edición de usuarios se ha creado el script *edit_user*.

Edición de grupos

Para la edición de grupos se siguen los mismos pasos que para la edición de usuarios. El formulario que aparece es el mismo que para dar de alta los grupos, salvo que incluye el botón 'Borrar registro'. Hay que advertir otra vez que si se borra un grupo se borraran todos los usuarios de ese grupo.

Cuando un grupo cambia de: tarifación, alta, baja, bono o tarifa hay que cambiar los datos de todos los usuarios que pertenezcan a ese grupo en la tabla *activacion*. Además si cambia de código de política habrá que cambiar el valor en la tabla *network* de la base de datos nocat.

Para los cambios en la tabla *activacion* hay que tener en cuenta dos casos:

1. Si el usuario no había entrado en un periodo activo. Si no ha entrado en un periodo activo, no hay nada registrado en el histórico aún. En este caso cambiamos los datos en activación y comprobamos si después del cambio está activo. Si es así, lo insertamos en nocat y creamos una fila en el histórico.

2. Si el usuario ya estaba activado. En este caso tendremos una fila creada en el histórico. Tenemos que cerrar el histórico, modificar los datos en *activacion* y comprobar si después del cambio está activo el usuario para meterlo en el histórico.

El script que se encarga de todo esto es *edit_group*.

Cambio de clave

La sección de cambio de clave está diseñada para cambiar la clave de los usuarios cuando ellos lo deseen, o en caso de pérdida. Para ello, se ha diseñado la sección de forma que no hace falta conocer la clave anterior. Simplemente se inserta la nueva clave dos veces y la clave anterior es sobreescrita. Esta sección está implementada en el script *cambio_clave*.

3.4.5. Consultas

Se han implementado dos tipos de consultas: consultas de usuarios y consultas de grupos.

Consultas de usuarios

Los parámetros de búsqueda son:

- Login.
- Apellidos.
- DNI.
- Grupo.
- Ubicación.
- Habitación.

Se puede ordenar por login, apellidos o ubicación y los datos que se obtienen pueden ser los datos personales del usuario, los datos de activación del usuario o ambos. Se da la posibilidad de elegir cuántos registros se quieren por página. El formulario de búsqueda de usuarios se muestra en la figura 3.18.

El resultado de una búsqueda en la que se muestran 10 registros por página, y están ordenados por apellidos está en la figura 3.19.

Consultas de grupos

Las consultas de grupo se pueden hacer por tipo de tarifación o buscar un grupo concreto. También se permite introducir el número de registros que se quiere que se muestren por página.

Desde ambas consultas se puede editar a un usuario o a un grupo pinchando con el ratón sobre él. Las consultas se guardan en un fichero de texto que puede ser descargado pinchando sobre un enlace que aparece bajo la consulta.

3.4.6. Informes

Se realizan dos tipos de informes: informe del número de conexiones, y un informe del consumo.

Figura 3.18: Consultas de usuarios

Figura 3.19: Consultas de usuarios

Informe del número de conexiones

Se introduce una fecha de inicio, una fecha de fin y una ubicación. El informe muestra una lista con el número de usuarios conectados en cada hora entre las 00:00:00 de la fecha de inicio y las 00:00:00 de la fecha de fin.

También se crea un fichero de texto que contiene las filas del informe separadas por tabuladores. Para bajarse o ver el fichero de texto hay un enlace bajo el informe.

El informe tiene el aspecto mostrado en la figura 3.20.

Figura 3.20: Informe de conexiones

Informe de consumo

Hay que introducir una ubicación, un login (opcional), una fecha de inicio, y una fecha de fin. El formulario que se presenta para rellenar tiene el aspecto de la figura 3.21.

Si no se introduce login se muestra un informe con todos los login y el consumo que han tenido entre la fecha de inicio y la fecha de fin. Si se introduce un login te muestra el consumo detallado de ese login. También se puede ver el consumo detallado de un usuario pinchando sobre él en el informe de todos los usuarios.

El informe de todos los usuarios se guarda en fichero de texto que está disponible para visualizar o guardar en un enlace. Un ejemplo de informe de consumo se puede ver en la figura 3.22.

Figura 3.21: Formulario para el informe de consumo

Figura 3.22: Informe de consumo

3.4.7. Facturación

El proceso de facturación incluye dos pasos: primero se muestra una factura de los conceptos que tengan el campo facturado igual a 'no' y segundo se cambia el campo de facturado a 'si' para dichos registros.

Para realizar la facturación se debe introducir un fecha hasta la que facturar. El programa se encarga de buscar todos los registros en las tablas *hco_activacion* y *hco_conexiones* que tengan el campo facturado igual a 'no' y su inicio sea anterior a la fecha de facturación. Una vez encontrados todos los registros que cumplan lo anterior se distinguen tres casos:

1. La fecha de fin es anterior a la fecha de facturación. Entonces se factura tal cual.

2. La fecha de fin es posterior a la fecha de facturación. En este caso se duplica la fila en el histórico. En la primera fila se pone como fecha de fin la fecha de facturación y el campo de facturado igual a 'si'. En la nueva fila que se ha creado se pone como inicio la fecha de facturación, como fin la fecha de fin que había en la fila inicial y el campo de facturado igual a 'no'. De esta manera queda facturado el periodo contemplado.

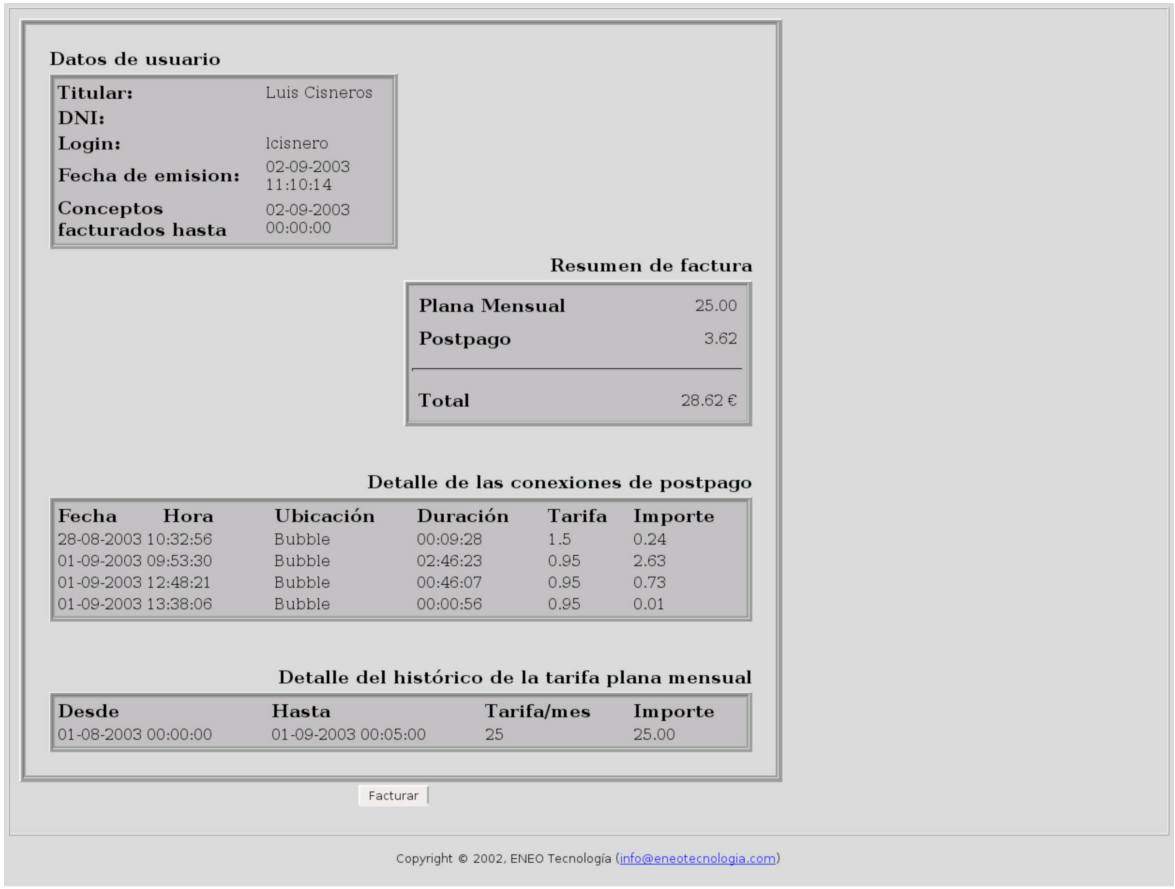
3. La fila tiene fecha de fin 'NULL'. En este caso quiere decir que el usuario sigue activo. Para estos casos se puede dar una doble casuística. Puede ser que la fecha de facturación sea mayor que el momento actual, o puede ser que la fecha de facturación sea anterior, pero el usuario siga activado. El primer caso no se va a contemplar, ya que no tiene sentido facturar hasta una fecha posterior al momento actual. En el segundo caso, se actuará de la misma manera que en el segundo punto. Es decir, se creará una nueva fila sin facturar con el periodo restante que seguirá teniendo el campo de fin igual a NULL y el campo facturado en 'no'.

Existen dos tipos de factura:

- Factura conjunta. La factura consiste en una tabla en la que cada fila corresponde a un login. El las filas se incluye el login de usuario, su nombre y apellidos, su DNI y el importe a pagar por ese usuario. Además al final de la factura aparece el total.

- Factura individual. La factura individual corresponde a un único login. En ella aparecen los datos de usuario, un resumen de la factura, y los detalles de activación y conexiones de ese login. En la figura 3.23 se muestra un ejemplo de factura individual.

El formulario de facturación incluye tres variantes:



Datos de usuario

Titular:	Luis Cisneros
DNI:	
Login:	lcisnero
Fecha de emision:	02-09-2003 11:10:14
Conceptos facturados hasta	02-09-2003 00:00:00

Resumen de factura

Plana Mensual	25.00
Postpago	3.62
Total	28.62 €

Detalle de las conexiones de postpago

Fecha Hora	Ubicación	Duración	Tarifa	Importe
28-08-2003 10:32:56	Bubble	00:09:28	1.5	0.24
01-09-2003 09:53:30	Bubble	02:46:23	0.95	2.63
01-09-2003 12:48:21	Bubble	00:46:07	0.95	0.73
01-09-2003 13:38:06	Bubble	00:00:56	0.95	0.01

Detalle del histórico de la tarifa plana mensual

Desde	Hasta	Tarifa/mes	Importe
01-08-2003 00:00:00	01-09-2003 00:05:00	25	25.00

Facturar

Copyright © 2002, ENEO Tecnología (info@eneotecnologia.com)

Figura 3.23: Factura detallada

1. Grupo. Si especificamos un grupo nos mostrará una factura conjunta de los usuarios de ese grupo.

2. Ubicación. Especificando una ubicación se mostrará una factura conjunta de los login que están en esa ubicación.

3. Login. Introduciendo el login de un usuario se muestra la factura individual de ese usuario. También se puede ver una factura individual desde una factura conjunta pinchando sobre un login determinado.

Una vez que se ha elegido el tipo de factura y ésta se ha mostrado por pantalla, aparece un botón de 'Facturar' debajo de la misma. Si el administrador está de acuerdo con los datos mostrados y quiere emitir la factura debe pulsar este botón, y en ese momento se cambian los campos facturado de las filas correspondientes.

El script que genera la página de facturación y procesa los datos se llama *facturacion*. Este script utiliza javascript para esconder el botón de facturar, después de facturar, y para preguntar confirmación antes de facturar.

3.4.8. Seguridad en el servidor

Autenticación

Para poder utilizar la aplicación de administración hay que introducir un nombre se usuario y una clave previamente. Se ha utilizado el proceso de autenticación de HTTP básico. La autenticación HTTP básica es muy sencilla. Cuando un explorador Web solicita un URL que está protegido por una autenticación HTTP, el servidor Web le devuelve una cabecera de estado 401 y otra de respuesta para la autenticación (véase figura 3.24). La cabecera contiene el sistema de autenticación que se está utilizando y el nombre del dominio. A continuación el explorador Web muestra un cuadro de diálogo que le pide al usuario que introduzca su nombre y contraseña. Cuando el usuario facilita esta información, el explorador se la envía de vuelta al usuario junto con el URL solicitado. El servidor comprueba si los datos son válidos. En caso, afirmativo, mostrará la página solicitada. En caso negativo, responde con un estado 401 y vuelve a enviar la misma cabecera de autenticación.

En la siguiente llamada al servidor, el explorador se limitará a enviar la información facilitada por el usuario, con lo que se evita que el servidor genere un estado 401. Por ejemplo, si el URL https://localhost/cgi-bin/admin/ requiere una autenticación HTTP básica, las siguientes llamadas a https://localhost/cgi-bin/admin/alta_usuario y https://localhost/cgi-bin/admin/activacion requerirán el par nombre/contraseña. Esta es la razón por la que el servidor envía el par antes de que se produzca ningún intercambio de autenticación (es decir, el envío de la cabecera de estado 401 y la respuesta de

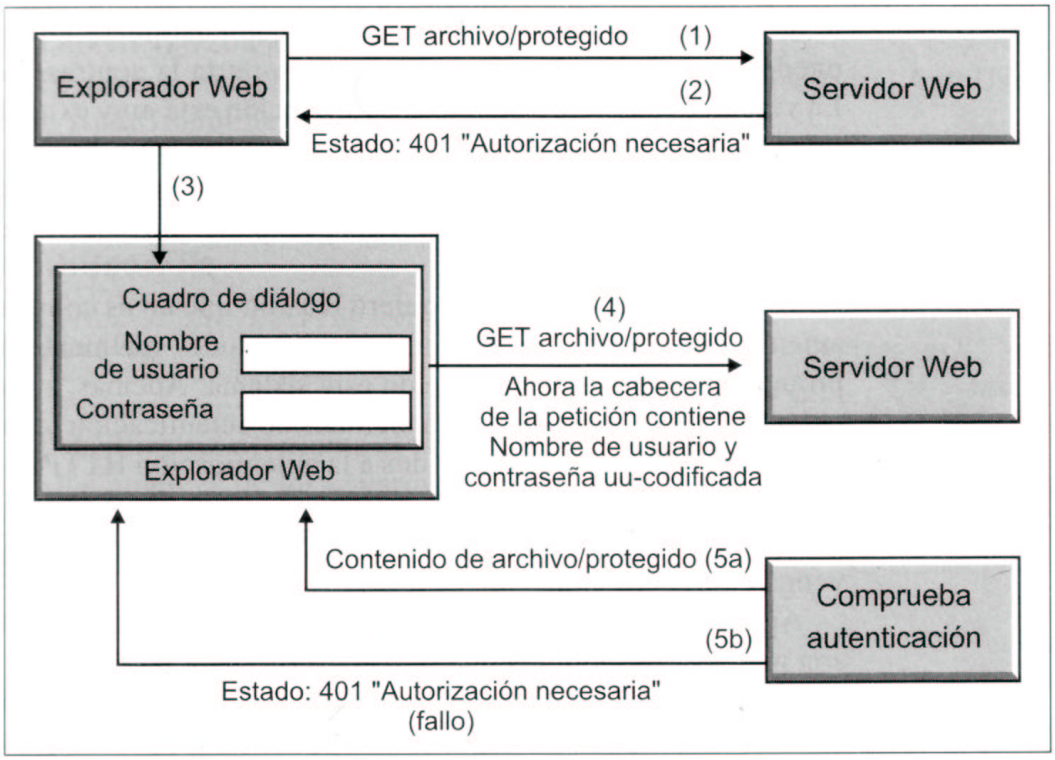


Figura 3.24: Proceso de autenticación HTTP básico

autenticación www) con el servidor. Este sistema es más rápido y práctico que generar una opción para cada petición recibida y hacer que el usuario introduzca una y otra vez su nombre y contraseña.

Cuando el sistema del cliente envía la contraseña (el host en el que se ejecuta el explorador Web), ni lo hace en texto plano ni la encripta. En vez de eso utiliza la codificación uu y la transmite por la red. Este es el inconveniente de este método de autenticación, porque cualquiera que tenga un sniffer puede hacerse con el paquete IP que transporta la contraseña uu-codificada. La verdad es que este sistema de codificación está muy extendido en la actualidad y cualquiera puede descodificar la contraseña y abusar de su uso. Es cierto que el sniffer tiene que localizar el paquete correcto para poder proceder a la descodificación, pero técnicamente es posible.

El módulo que se ha usado para proteger los scripts cgi de un usuario externo es mod_auth. Este módulo se compila por defecto en la distribución estándar de apache. Para confirmar la autenticación, utiliza los nombre de usuarios, los grupos y las contraseñas almacenadas en archivos de texto. Este sistema funciona muy bien cuando el número de usuarios con el que se trabaja es muy pequeño. En nuestro caso la aplicación sólo la utilizará el administrador del sistema.

En nuestro caso vamos a crear un único usuario por lo que sólo necesitaremos la directivas AuthUserFile.

La directiva AuthUserFile determina el nombre del archivo de texto en el que se guardan los nombres de los usuarios y las contraseñas que se utilizan en el proceso de autenticación HTTP básico. Se ha de suministrar un path completo.

La configuración de Apache que tenemos tiene las siguientes directivas:

```
ScriptAlias /cgi-bin/ /usr/local/nocat/cgi-bin/
AccessFileName .htaccess
AllowOverride All
```

Nosotros queremos restringir el acceso al directorio /usr/local/nocat/cgi-bin/admin. Los pasos que se han seguido son:

1. Crear un archivo para el usuario con htpasswd. Para esto se he utilizado un programa llamado htpasswd que incluye la distribución estándar de Apache. El programa crea el archivo del usuario que necesita la directriz AuthUserFile. El comando utilizado es:

   ```
   htpasswd -c /usr/local/nocat/usuarios/.htpasswd nocat
   ```

   "nocat" es el nombre de usuario que hemos introducido. Seguidamente la utilidad htpasswd solicita la contraseña de "nocat".

   Hay que destacar que:

   - La opción -c es para crear un nuevo archivo.
   - El archivo está colocado fuera del árbol de directorios de apache, para que nadie pueda copiarlo desde la red.
   - Se ha utilizado un punto delante del nombre para que no aparezca en la salida del sistema.

2. Creación del archivo .htaccess. Hemos creado el archivo /usr/local/nocat/cgi-bin/admin/.htaccess con los siguientes datos:

   ```
   AuthName "Sistema de Administración"
   AuthType Basic
   AuthUserFile /usr/local/nocat/usuarios/.htpasswd
   require user nocat
   ```

   La primera directriz, AuthName, determina la extensión de la autenticación. No es más que una etiqueta que se envía al explorador Web para que tenga alguna

pista sobre a qué se quiere acceder. En este caso, "Sistema de administración". La segunda directriz, AuthType, especifica el tipo de autenticación utilizada. La siguiente directriz, AuthUserFile, especifica el nombre del archivo del usuario.

3. Configurar el archivo de permisos. Hay que cambiar el propietario de los archivos .htaccess y .htpasswd para que sólo los pueda leer Apache.

4. Cuando intentamos acceder a cualquier página dentro del directorio /cgi-bin/admin nos aparece un cuadro de diálogo similar al de la figura 3.25.

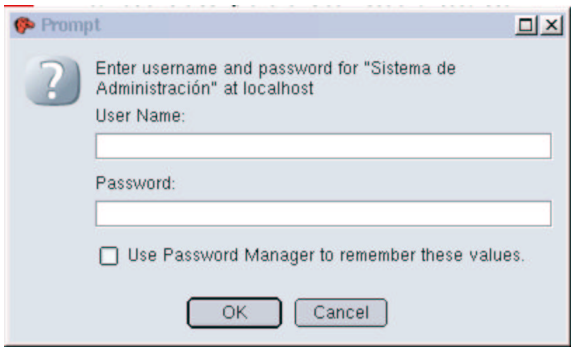


Figura 3.25: El explorador Web solicita el nombre y la contraseña

Seguridad en el servidor

Vamos a contemplar tres aspectos básicos de la seguridad en el servidor:

- Mantener la integridad de la información.
- Evitar que se utilice nuestro servidor Web como punto de infiltración a la red de la organización.
- Evitar que nuestro servidor Web se use como punto de paso de las intrusiones dirigidas a otras redes.

Para conseguir estos tres aspectos nos hemos centrado en evitar errores de programación en los CGI. Además de estas medidas, que se comentarán en el siguiente punto, habría que tener en cuenta varios puntos de control adicionales:

- Red de trabajo. Es conveniente aislar el servidor Web de la red local de trabajo. Esto se consigue poniendo un firewall. De esta manera los intrusos no podrán acceder a máquinas de la red interna. Otra forma es poner el servidor Web detrás de un firewall.

- Software del servidor Web. Es conveniente para la seguridad configurar todas las directivas de Apache de forma que sean lo más restrictivas posible. Por ejemplo, las directivas User y Group no deben ser nobody y nogroup, deben ser el nombre del usuario que podrá ver y ejecutar los documentos y programas del servidor. Se deben proteger los directorios ServerRoot y los directorios de registro de forma que nadie pueda escribir en ellos. También es recomendable desactivar el acceso predeterminado.

Seguridad en los CGI

Se han tenido en cuenta tres aspectos y se han intentado evitar los siguientes riesgos:

- Perdida de información. Se ha tratado de evitar que un hacker pueda descubrir nuevos métodos para acceder al sistema.
- Ejecución de comandos del sistema a través de la aplicaciones CGI.
- Consumición de los recursos del sistema.

Control de la entrada de datos

El principal peligro de los CGI proviene de la entrada de datos en los formularios. Se pueden producir fallos tanto por la introducción de datos maliciosos de forma premeditada, o bien, por una equivocación al introducir los datos que provoque una acción inesperada.

- Evitar que un programa se bloquee por la entrada de datos. Un usuario podría introducir en un campo de texto un documento de varios megas lo que provocaría que se saturase el servidor y causaría un error en la base de datos. Para evitar esto en todos los formularios se controla que los campos de texto tengan la cláusula maxlength que te limita la longitud de la entrada.
- Evitar que la entrada del usuario haga entradas al sistema que no sean seguras. Para evitar esto se utilizan dos medidas. Comprobar todas las entradas del sistema. Todos los datos que se introducen son pasados por una expresión regular que controla que los datos introducidos se ajustan a los caracteres permitidos. De esta manera se evita que los usuarios puedan meter metacaracteres en los campos para ejecutar alguna llamada la sistema.

  En nuestro caso nos protegemos de que se introduzca código dentro de las sentencias SQL. Esto se llama "SQL injection" y se consigue insertando comillas

simples dentro de las sentencias. Con la expresiones regulares vistas anteriormente se comprueba que no se permita el uso de comillas simples en nuestros formularios.

Además de comprobar los datos introducidos se utiliza la opción -T en la línea en la que se llama al interprete de Perl. La opción -T hace que Perl localice las variables ocultas y no permita el uso de las funciones system(), exec(), piped(), open() o eval(). Cualquier variable que no vaya a utilizar el programa (incluyendo las variables de entorno, STDIN y las procedentes de la línea de comandos) se considera oculta y no se utilizará.