Examen de Introducción a las Redes de Computadoras
(ref: sirc0208.doc)

30 de julio de 2002

***Atención**: para todos los ejercicios, suponga que dispone de los tipos de datos basicos (p.ej. lista, cola, archivo, string, etc.) y sus funciones asociadas (ej: tail(lista), crear(archivo), concatenar(string, string).*

Ejercicio 1

Se quiere diseñar un protocolo para transferir un archivo desde un conjunto de servidores hacia un cliente. Se dispone de un servicio de capa 4 sin conexión (UDP). La transmisión del archivo se hará en partes, pero se transmitirá un archivo únicamente; es decir, el archivo se secciona y se bajan las distintas partes desde distintos servidores.

Los archivos se identifican con un nombre y un tamaño que se supondran únicos. Cada archivo se divide en bloques de 512 bytes, para facilitar la transmisión via UDP. El cliente obtiene de la capa superior el nombre y el tamaño (en bytes) del archivo solicitado mediante la función

`archivo(string& nombre, int& tamano)`.

Este debe solicitar la transferencia de los bloques al máximo número de servidores posibles, a fin de minimizar el tiempo de transferencia.

Para no sobrecargar ningún servidor, se solicitarán el mismo numero de bloques a cada uno.
Se dispone de una lista (con las direcciones IP) de servidores de archivos conectados, que se obtendrá mediante la funcion

```
Lista obtenerServidores()
```

Cada servidor acepta solicitudes de un único cliente en el puerto CON.
Se debe tener en cuenta que no todos los servidores disponen de todos los archivos.
Además, al estar basado en un protocolo sin conexión, se debe considerar el caso en que se pierdan paquetes (sin embargo los paquetes *siempre* llegan en orden).

Una vez que el cliente determina la lista de bloques a obtener de cada servidor solicita estos para reconstruir el archivo. Cuando se han obtenido *todos* los bloques, se dispone de las siguientes funciones para guardarlo en disco:

```
Archivo concatenar(Archivo ar, Bloque bl)
Grabar(Archivo ar, string nombre)
```

Se pide:

- Diseñar la interfaz de comunicación entre los servidores y el cliente, y las estructuras necesarias.
- Especificar en un lenguaje de alto nivel el proceso que se ejecuta en el cliente.

Solución:

Se implementara un protocolo basado en UDP con paquetes de diferentes tipos para la comunicación entre los Servidores y el Cliente.
Los datos a intercambiar son el nombre del archivo a transmitir, el numero de bloque que se transmite – que se envia como multiplo de 512 bytes– y los datos del bloque. Ademas se pueden perder paquetes por lo que se debe implementar un algoritmo para evitarlo, por lo cual eligiremos Stop&Wait. En este caso hay N comunicaciones por lo que se debe mantener un array de secuencias y un array de confirmaciones y debemos agregar un campo de secuencia y confirmación a cada paquete.
El formato de paquete elegido es el siguiente

Tipo : {consulta, respuesta, confirmacion, pedido, datos, fin}	=>Indica el tipo de paquete que se envia
Archivo : string	=>El nombre del archivo para la consulta, pedido, etc
Bloque : integer	=>El numero de bloque que se pide, responde, etc
Datos : bit[]	=>El bloque de datos transmitido
Respuesta : {V, F}	=> V – el servidor tiene el archivo; F – no lo tiene
Secs : {0,1}	=>Numero de secuencia para Stop&Wait
Conf : {0,1}	=>Numero de confirmacion para Stop&Wait

El servidor aceptara paquetes de tipo consulta, pedido y fin; el cliente aceptara paquetes de tipo respuesta, datos y fin.
A continuación describimos la logica del protocolo a utilizar.

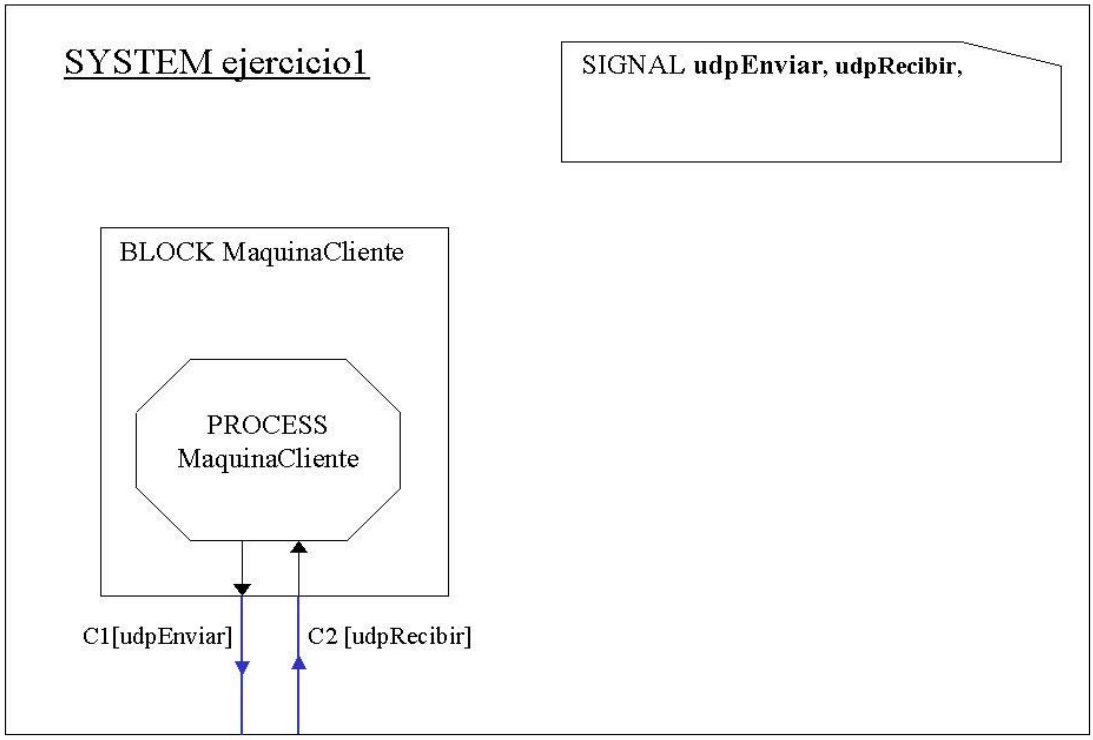
CLIENTE

1. Se obtienen los servidores activos, el nombre de archivo y el tamaño
2. Se envia un paquete de tipo “consulta” a cada servidor activo
3. Se espera la recepción de un paquete de tipo “respuesta” o el vencimiento del tiempo de espera
   - Si la respuesta es V se agrega el servidor a la lista de servidores disponibles y se envia un paquete de tipo “confirmacion”
   - Si se vence el tiempo de espera se reenvia el paquete de tipo consulta
4. Si se vence el tiempo de espera total se asume que todos los servidores que no contestaron no tienen el archivo
5. Se envia un paquete de tipo pedido a cada servidor disponible ordenado segun el numero de bloque
6. Se espera la recepción de un paquete de tipo “datos” con los datos del bloque que se almacenan para luego reconstruir el archivo y se pide el siguiente bloque al servidor.
7. Si se obtuvieron todos los bloques se envia un paquete de tipo “fin”
8. Se espera la recepción de un paquete de tipo “fin” o el vencimiento del tiempo de espera
   - si se vence el tiempo de espera o el tipo de paquete no es “fin” se reenvia el paquete de tipo fin
9. Se reconstruye el archivo con los datos recibidos y se guarda el archivo.

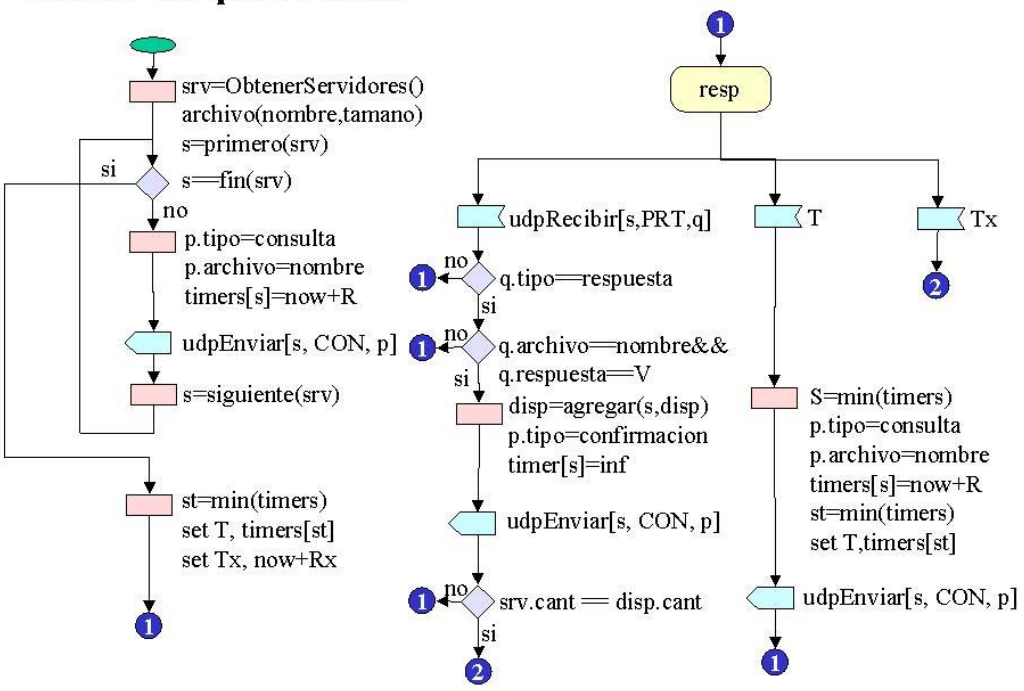
SERVIDOR

1. Espera la recepcion de paquetes
2. Si se recibe un paquete de tipo consulta se envia un paquete de tipo “respuesta” con el campo respuesta en V si se tiene el archivo
3. Se espera por un paquete de tipo “confirmacion”, si no se recibe se reenvia el paquete de tipo “respuesta”
4. Se espera la recepcion de un paquete de tipo “pedido” y se envia un paquete de tipo “datos”
5. Se espera la confirmacion del paquete, si se vence el tiempo de espera se reenvia el paquete de tipo “datos”

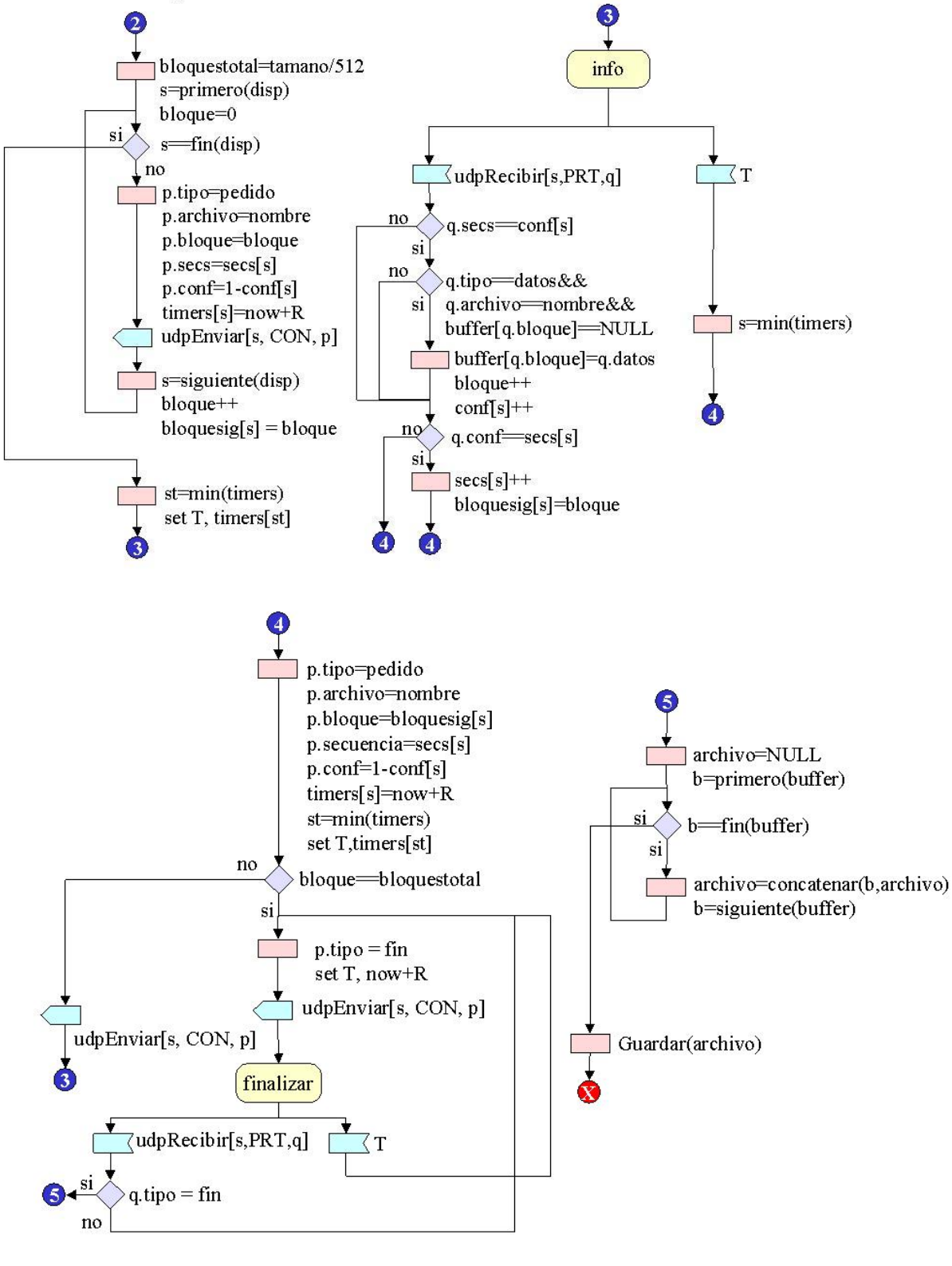
Si en algun momento se recibe el paquete de tipo “fin” se envia un paquete de tipo “fin”.



Process MaquinaCliente

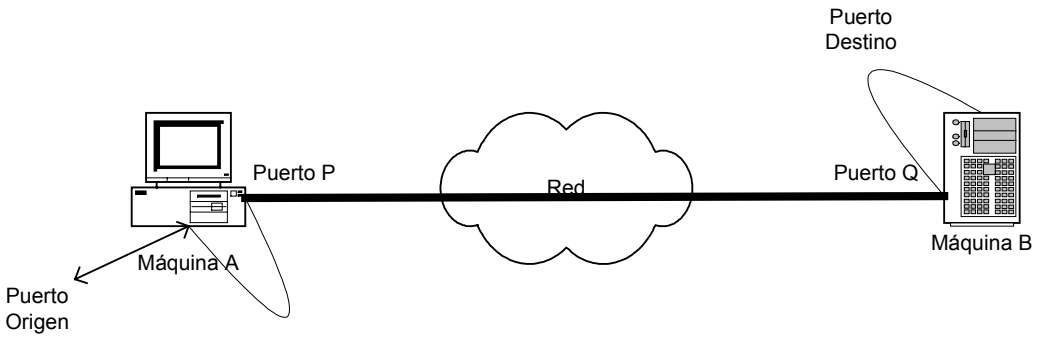


Process MaquinaCliente



Ejercicio 2

Una máquina A desea establecer una conexión con una máquina B, mediante la técnica conocida como *port forwarding*. Esto se ejemplifica en la siguiente figura:



La máquina A tiene un proceso esperando conexiones (pasivo) en un puerto P. El proceso, luego de recibir una conexión, establece una conexión activa con la máquina B en el puerto Q.
La máquina B recibe una conexión pasiva de la máquina A en el puerto Q y posteriormente se encarga de hacer la conexión en la misma máquina B con el puerto destino.

p.ej: si se desea hacer un *port forwarding* de un servicio de chequeo de mail via POP3 (puerto 110 TCP), se haria lo siguiente:

Se inicia el servicio de *port forwarding* entre ambas máquinas, estableciendo el túnel de conexión entre puerto local 5678 y la maquina *pop3server* puerto 4455. Luego el servicio en la máquina *pop3server* se encargaría de hacer la conexión con el puerto 110 y de enviar los datos hacia ambos lados. Si alguien quiere hacer un chequeo de mail, se conectará localmente al puerto 5678, en vez de al servidor *popserver*.

Vista la descripción del *port forwarding* se pide:

a) Especifique en un lenguaje de alto nivel un servicio que se encargue de hacer el port forwarding entre dos máquinas (entre el puerto P de la máquina A y el puerto Q de la máquina, se supone aplicación de destino implementada).

b) La técnica anterior puede utilizarse, con una pequeña modificación, para tener comunicaciones *seguras* entre dos máquinas (mediante criptografia). Sugiera y especifique una modificacion a la parte a), en la cual pueda establecerse una comunicacion segura entre las máquinas. Puede utilizar las primitivas sig:

- `encrypt(string, key).`
- `decrypt(string, key).`

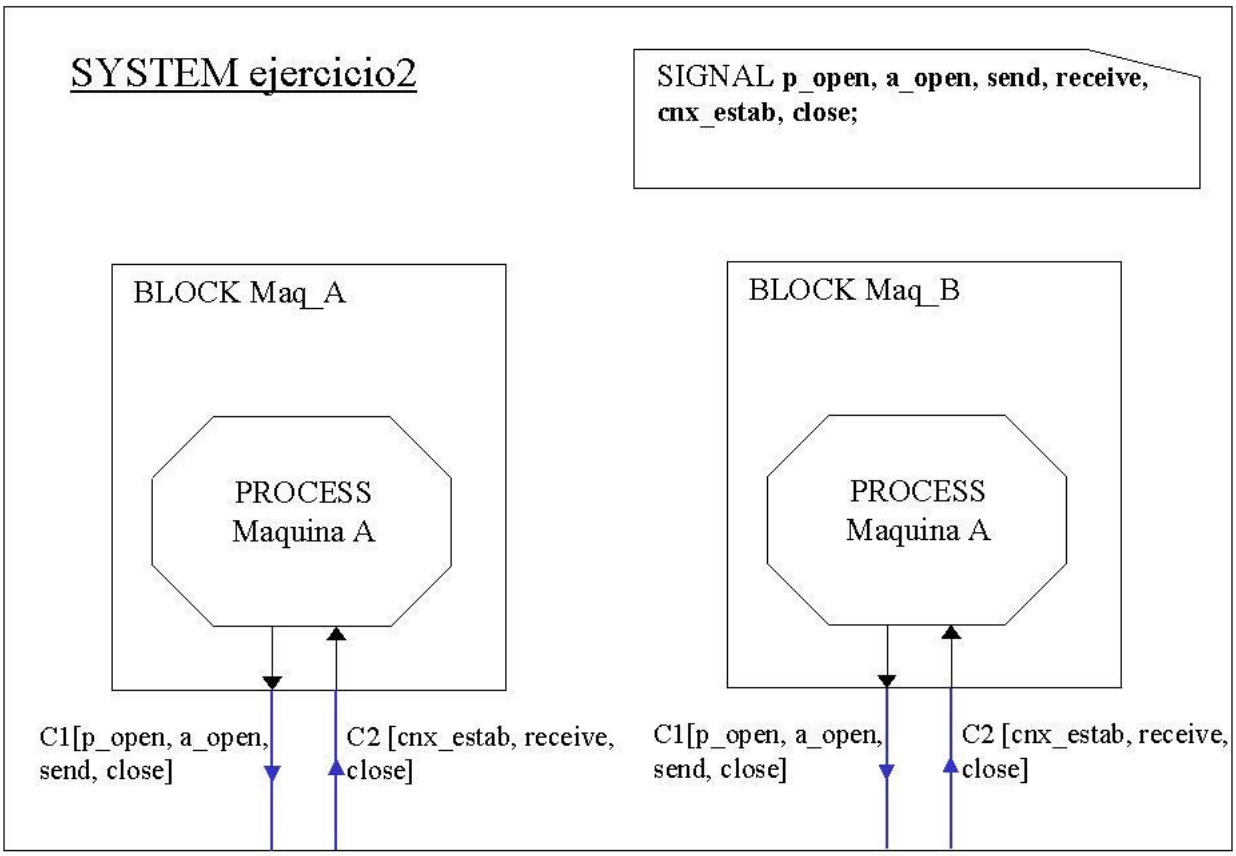
Atención: sólo incluya la especificación de la modificación, no repita el problema de la parte a)

Solución:

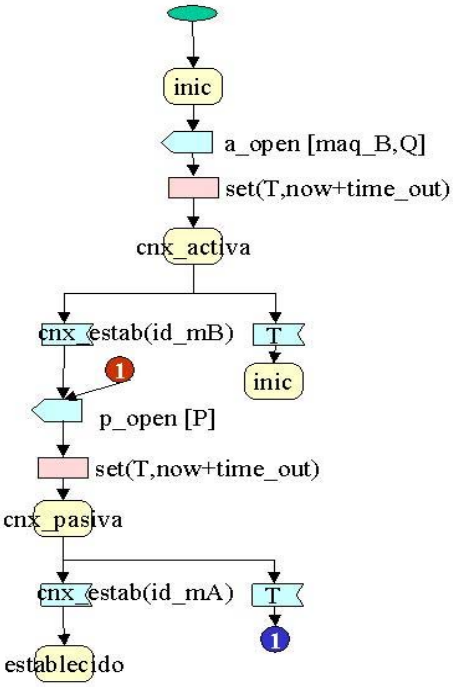
Parte a)

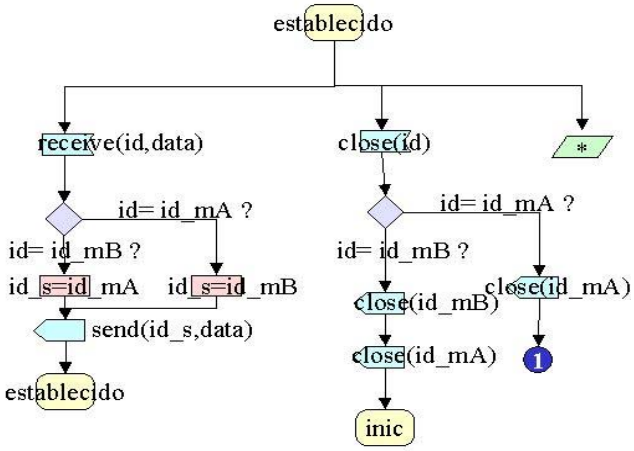
Se utilizarán primitivas de TCP/IP, a_open, p_open, cnx_estab, send, receive y clos..

Se supone que la máquina A atiende una conexión por vez, y cuando esta se cierra establece una nueva y envía los datos por el canal establecido. Conjuntamente se supone que el servicio atendido (en el caso de ejemplo POP3), no requiere establecer una nueva conexión cada vez que se autentica un nuevo usuario.

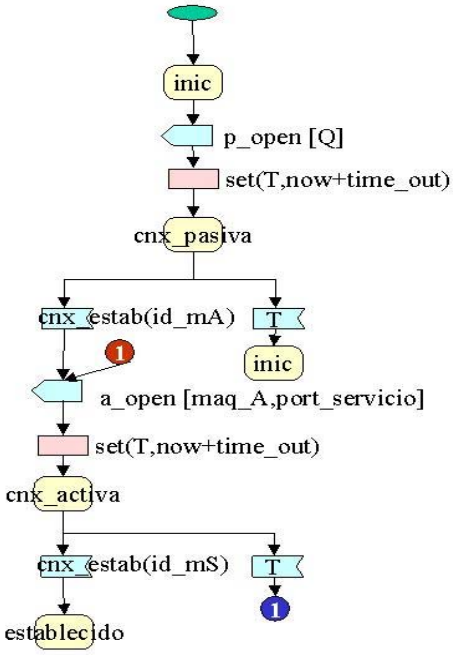


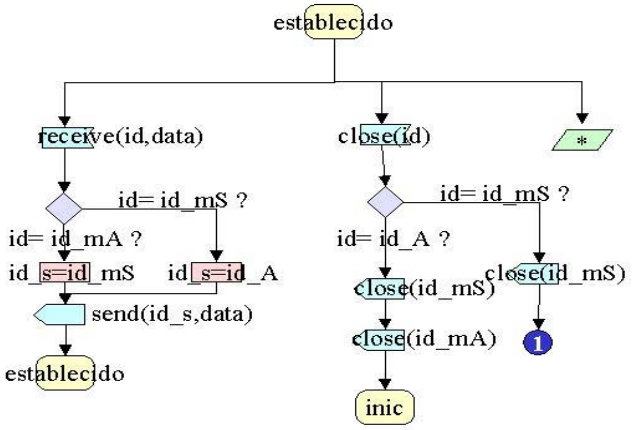
Process Maquina_A





Process Maquina_B



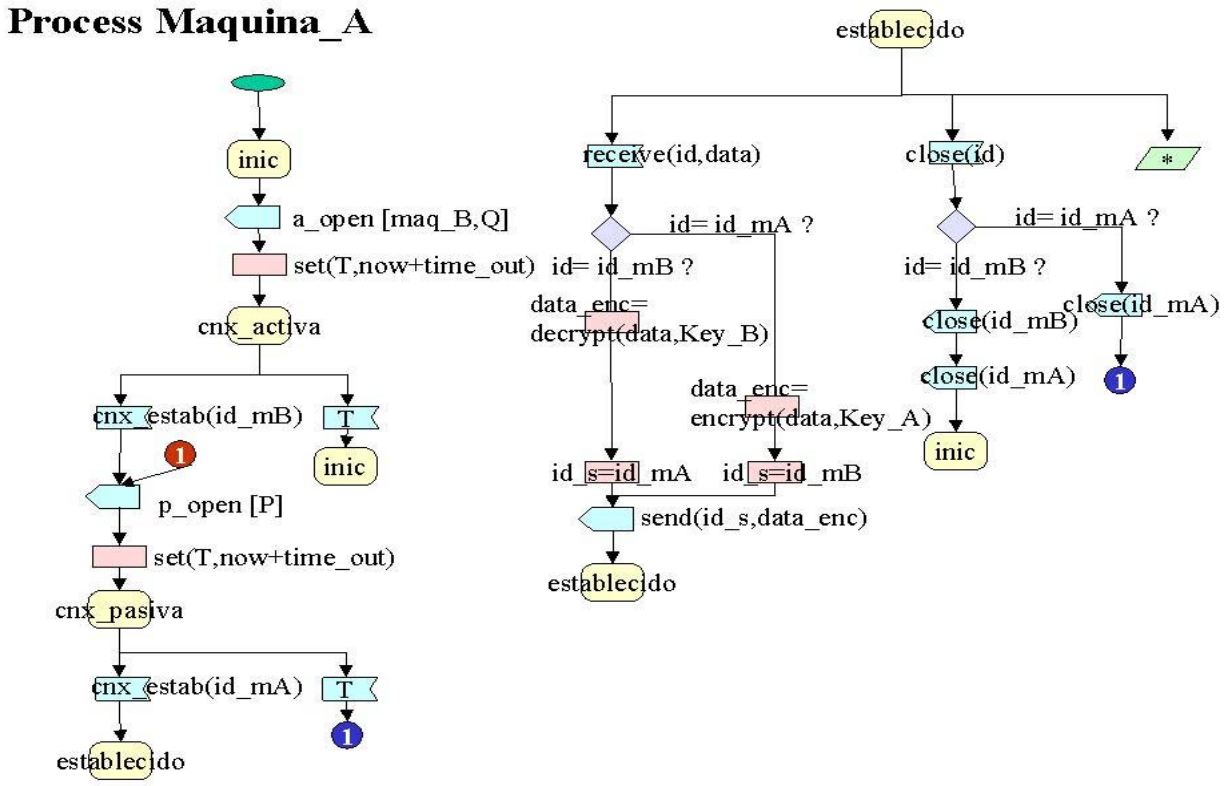


Parte b)

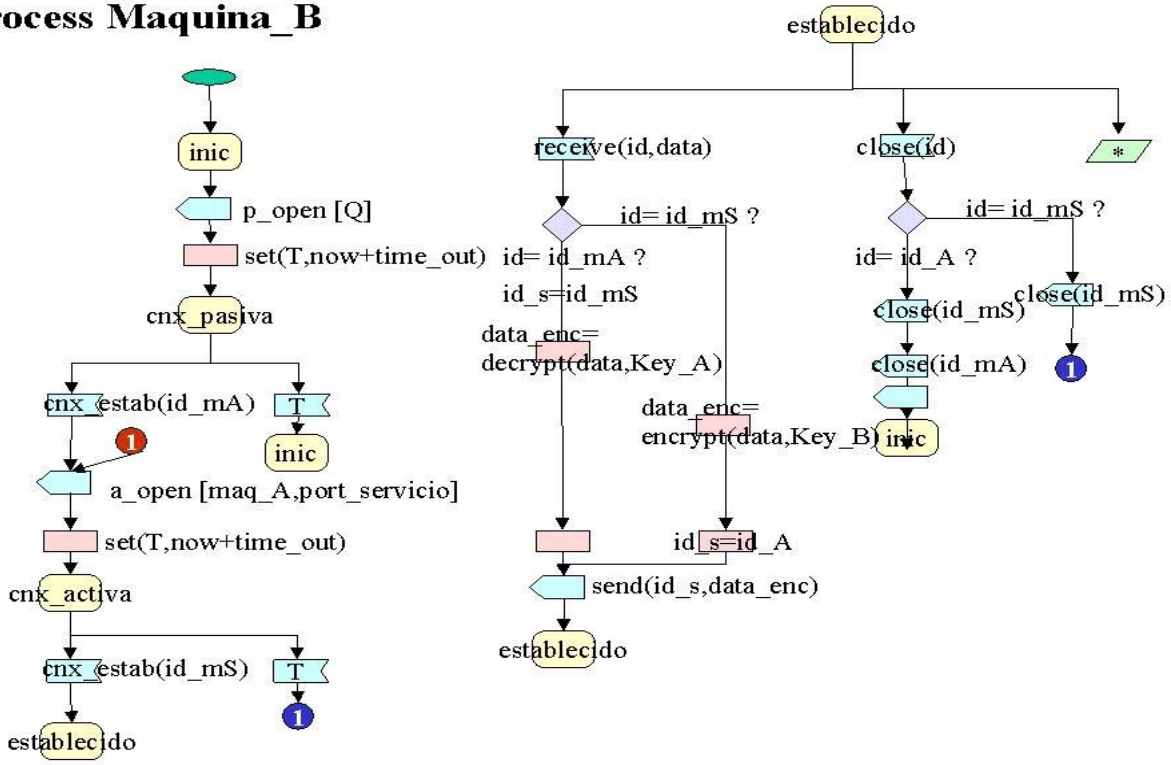
Para mantener la seguridad y evitar que se visualice los datos transmitidos por medio de un análisis de los paquetes que intercambian las máquinas, se puede utilizar un encriptado en la conexión entre Maquina_A y Maquina_B.

Supondré que cada equpo conoce la clave de encripción del otro (Key_A, y Key_B), y que con las primitivas sugeridas se encripte la parte de datos del paquete.

El System no varía, lo que se presentará son los Process Maquina_A y Maquina_B, con el agregado de encriptado del mismo.



Process Maquina_B



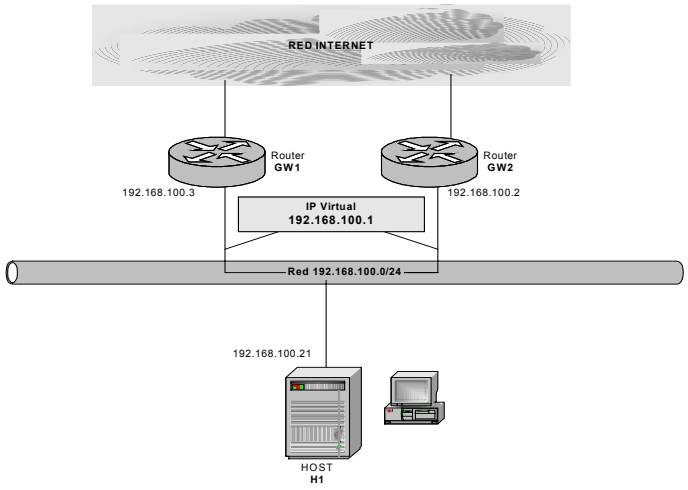
Ejercicio 3

Se considera la siguiente red de computadoras con conexión a la Internet.

El host H1 tiene configurada una ruta por default completamente estática, y no es capaz de correr ningun protocolo de enrutamiento dinámico (ni RIP, ni OSPF, etc.).

A los efectos de proporcionar redundancia a la salida hacia Internet de esta red, se instalan dos routers, GW1 y GW2, para que en caso de la falla de uno de ellos, el otro pueda tomar la funcionalidad del otro.

Se asume que el enrutamiento desde la red Internet hacia la red en estudio esta resuelto.



Se pide:

1. En el caso de que se configure como ruta por defecto en H1 las direcciones IP de GW1 o de GW2 (NO es posible configurar ambos), ¿que ocurre si el router en uso falla?. ¿por qué?

2. A los efectos de solucionar este problema, se plantea el siguiente protocolo para poder aprovechar a ambos routers. En H1 se configurará una ruta por defecto apuntando a una *direccion IP virtual*, la 192.168.100.1 en el diagrama, la cual podrá ser atendida por uno de ellos (el MASTER) y en caso de falla del master, por el secundario. Se asumirá que GW1 es el master y GW2 el secundario. Se dice que GW1 y GW2 forman un *grupo*.

> 2.1 Describa el funcionamiento a nivel de capa 2 y 3 del protocolo. Estudie el funcionamiento normal y el funcionamiento en caso de falla del master. (*Se puede utilizar el protocolo por el cual se mapean direcciones de capa 2 y de capa 3)*.
>
> 2.2 A los efectos de controlar la toma de la IP virtual por GW1 o GW2, es necesario un protocolo de control entre ambos. Se propone implementar un protocolo sobre UDP, al cual se configuran estaticamente los miembros del grupo. El master enviará al otro miembro del grupo paquetes periódicamente (*keepalives*) reportando su estado y asegurando que está activo. El secundario deberá tomar el control sobre la dirección IP virtual en la medida de que se pierdan más de *N* keepalives. Deberá devolver el control al primario si este vuelve a la vida y comienza a enviar keepalives nuevamente.

Se pide especificar este protocolo en algun lenguaje de alto nivel.

Solución:

1. El funcionamiento en forma resumida es el siguiente, al equipo se le configura un equipo como default gateway, por medio de la IP que el equipo tiene en la LAN a la que esta conectado.

A través de ARP el host H1 resolverá la dirección hardware (ethernet) hacia donde enviará los paquetes que no tenga ruta por defecto.

Ante la falla del router, el host H1 continuará intentando enviar los paquetes a la dirección ethernet resuelta a través de ARP, sin lograr salir a Internet.

2.
2.1 Para la solución se propone el siguiente protocolo:

Se considerará que existe un equipo *master*, que responde los requerimientos de la red. Esto supondrá resolver, ya sea recepción de paquetes a ser enviados hacia Internet, o resolución de direcciones de capa 2 y capa 3, donde se utilizará ARP.

Así mismo el equipo *respaldo* continuamente verificará la recepción de *keepalives*, verificando que la dirección IP virtual se encuentre activa. En caso de falla el equipo de RESPALDO tomará la respuesta de la dirección IP virtual.

Se utilizarán para esto datagramas con el siguiente formato.

```
struct datagrama{
      char tipo;         /* El tipo contiene un byte según el siguiente código
                              0x01  Keepalive (KA)
                              0x02  Keepalive Request (KA_req)
                              0x03  Bajar interfase (down_req)
                              0x04  Interfase baja OK (down_OK) */
      char data[125];
}
```

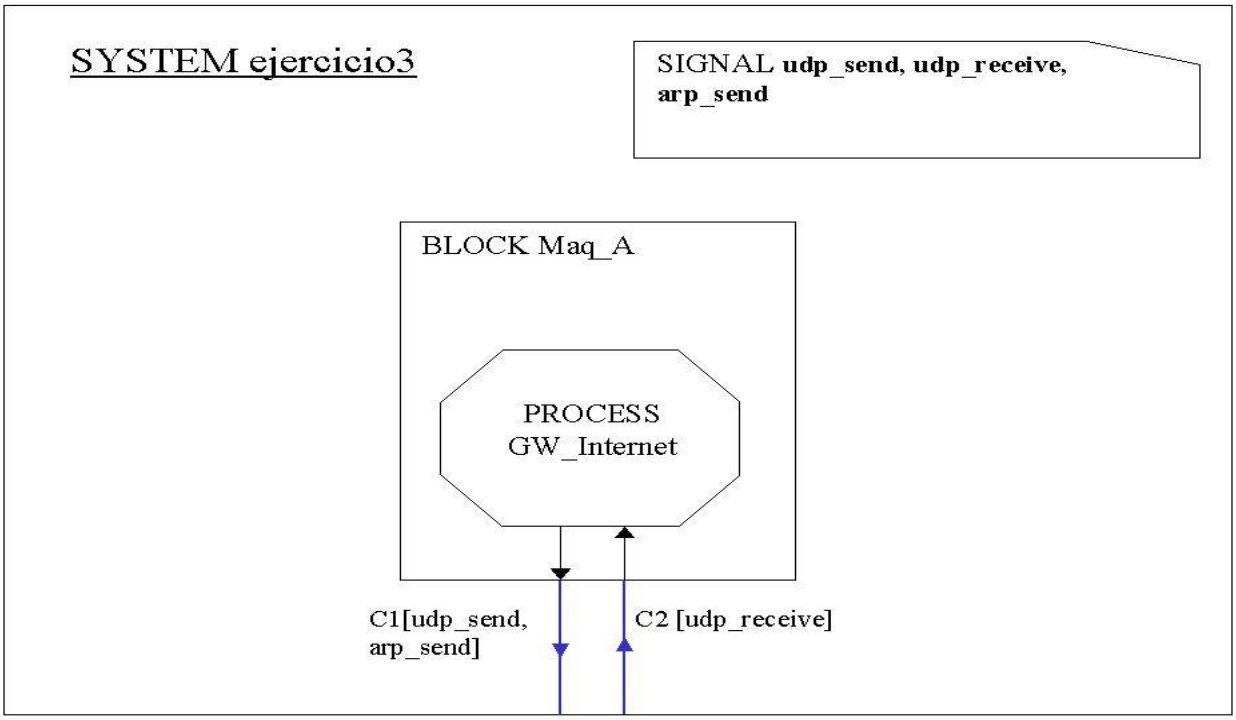
2.2 La solución propuesta supone la existencia de dos equipos, el *master*, que es quien tiene activa la IP virtual definida como default gateway de los equipos que integran la LAN, y otro equipo *respaldo* que controla el funcionamiento y en caso de falla solicita la utilización de la IP virutal.

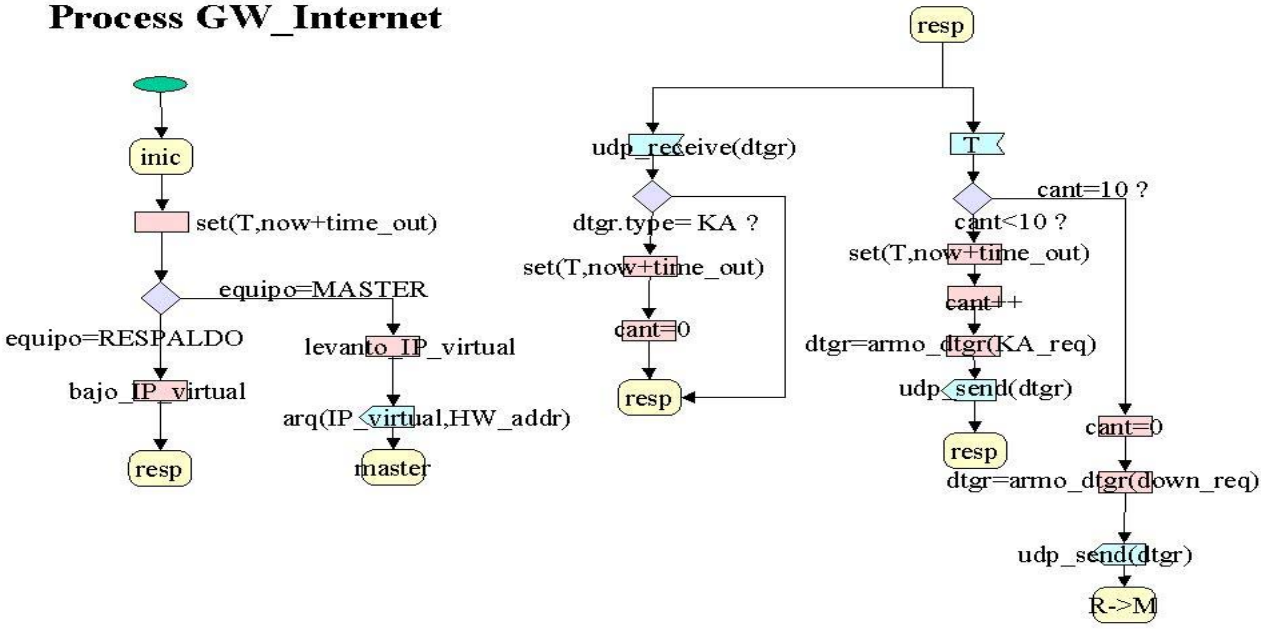
Como se puede ver en la solución planteada en el SDL presentado más adelante, el funcionamiento del protocolo se resume en:

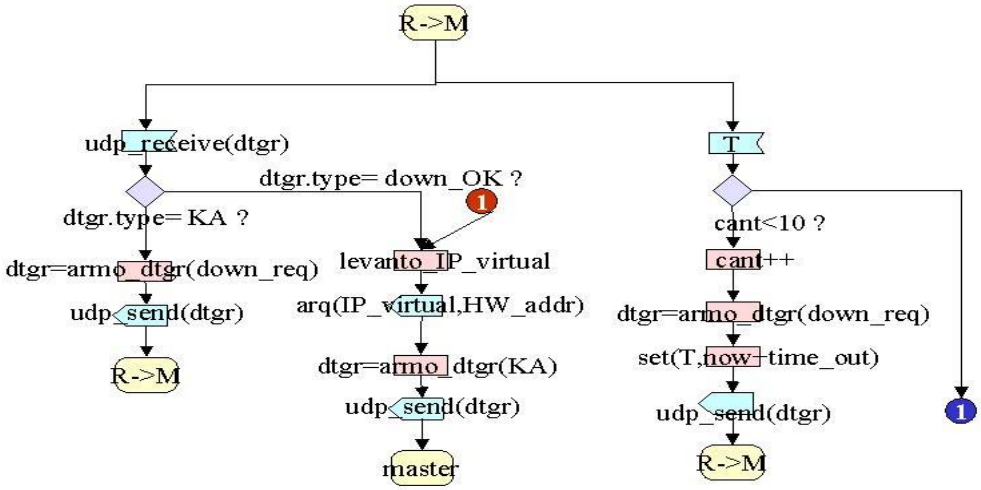
- El equipo *master* envía cada un tiempo predefinido, un paquete tipo KA, que indica que el equipo esta funcionando en forma correcta.
- El equipo *respaldo*, controla la recepción de los paquetes tipo KA provenientes del equipo *master.* De no recibir en primera instancia envía un paquete tipo KA_req,
- El equipo *master* ante la recepción de un paquete tipo KA_req responde inmediatamente con un paquete tipo KA.
- Si en un tiempo de 10 time_out no se recibe un paquete tipo KA, entonces el equipo *respaldo*, solicita la baja de la IP virtual al equipo *master.* Esto se realiza a través de un paquete tipo down_req. El equipo *master*, ante la solicitud de un down_req, baja la IP virtual y devuelve un paquete tipo down_OK. De no conseguirse la respuesta después de vencido un tiempo de 10 time_out el respaldo toma la IP virtual.
- Para notificar en la LAN el cambio de dirección ethernet de la IP virtual, envía un paquete ARP.
- El protocolo como se sugiere se comunica a través de datagramas UDP.

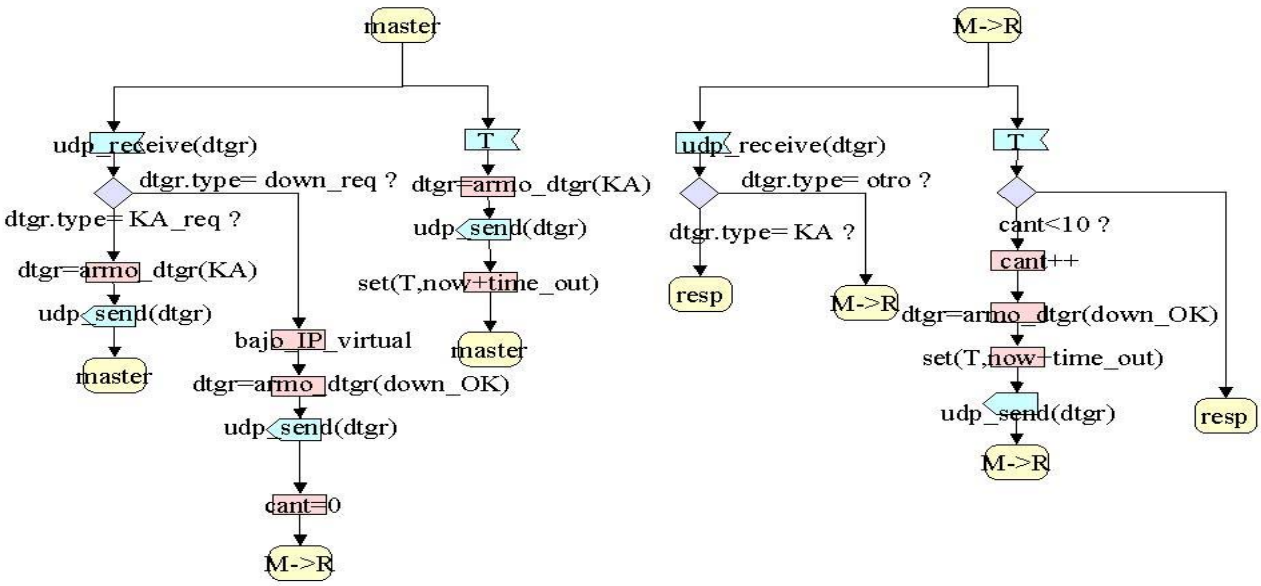
Observaciones sobre la solución propuesta:

- Es aconsejable que el intercambio de información del protocolo se realice por otra interface de red, para que el protocolo tenga independencia de la LAN que tiene la IP virtual.
- Pueden existir equipos que ante un ARP con cambio de dirección IP interprete que la IP se encuentra duplicada en la LAN. La única solución al problema sería que los equipos GW_internet, modificasen su dirección ethernet recibiendo los paquetes para la IP default gateway




Process GW_Internet
inic
set(T,now+time_out)
equipo=MASTER
equipo=RESPALDO
bajo_IP_virtual
resp
levanto_IP_virtual
arq(IP_virtual,HW_addr)
master
resp
udp_receive(dtgr)
dtgr.type= KA ?
set(T,now+time_out)
cant=0
resp
T
cant=10 ?
cant<10 ?
set(T,now+time_out)
cant++
dtgr=armo_dtgr(KA_req)
udp_send(dtgr)
resp
cant=0
dtgr=armo_dtgr(down_req)
udp_send(dtgr)
R->M


R->M
udp_receive(dtgr)
dtgr.type= down_OK ?
dtgr.type= KA ?
dtgr=armo_dtgr(down_req)
udp_send(dtgr)
R->M
levanto_IP_virtual
arq(IP_virtual,HW_addr)
dtgr=armo_dtgr(KA)
udp_send(dtgr)
master
T
cant<10 ?
cant++
dtgr=armo_dtgr(down_req)
set(T,now+time_out)
udp_send(dtgr)
R->M


master
udp_receive(dtgr)
dtgr.type= down_req ?
dtgr.type= KA_req ?
dtgr=armo_dtgr(KA)
udp_send(dtgr)
master
bajo_IP_virtual
dtgr=armo_dtgr(down_OK)
udp_send(dtgr)
cant=0
M->R
T
dtgr=armo_dtgr(KA)
udp_send(dtgr)
set(T,now+time_out)
master
M->R
udp_receive(dtgr)
dtgr.type= otro ?
dtgr.type= KA ?
resp
M->R
T
cant<10 ?
cant++
dtgr=armo_dtgr(down_OK)
set(T,now+time_out)
udp_send(dtgr)
M->R
resp