Práctica 1 (TSR)

Sesiones 1, 2 y 3

Sistema de máquinas virtuales del DSIC (portal-ng.dsic.cloud)

- Las máquinas virtuales tienen el nombre genérico:

  tsr-XXX-prueba0.dsic.cloud

- Cada alumno tiene un identificador **XXX** asignado.
- Por tanto, cada alumno tendrá una máquina virtual.
- El acceso a la máquina virtual requiere tener instalada una **VPN**.
- Si accedemos a través de **windesktop.dsic.upv.es** o **linuxdesktop.dsic.upv.es** ya está la **VPN** instalada (será el método más sencillo y habitual).
- La máquina virtual se gestionará a través del enlace: **portal-ng.dsic.cloud** (se requiere estar en linuxdesktop)

Sistema de máquinas virtuales del DSIC (portal-ng.dsic.cloud)

- Tendremos derechos de superusuario (usuario **root**) sobre nuestra máquina virtual (conviene crear otro usuario para evitar situaciones comprometidas).
- Nuestra máquina virtual tendrá un sistema de ficheros independiente y no tendrá sistema de fichero compartidos (no podremos acceder al **disco w**).
- VENTAJAS:
  - Disponemos de puertos independientes para nuestras aplicaciones.
  - Las máquinas virtuales **tsr-XXX-prueba0.dsic.cloud** pueden verse unas a otras.
  - **De esta forma podemos desplegar nuestras aplicaciones distribuidas emulando un sistema distribuido real.**

Acceso a nuestra máquina virtual
(tsr-XXX-prueba0.dsic.cloud)

- Desde los desktops del laboratorio accedemos a la URL:

  https://portal-ng.dsic.cloud

- Añadimos las excepciones necesarias para que nos permita entrar…
- Y nos identificamos con nuestra clave del DSIC.
- Aparecerán la lista de máquinas virtuales que tenemos asignadas..
- Y ponemos en marcha **tsr-XXX-prueba0.dsic.cloud**
- NOTA: desde portal-ng podemos consultar la dirección IP de nuestra máquina.
- FORMAS DE ACCESO desde un terminal de **linuxdesktop** (usr: **root**)
  - `$ ssh tsr-XXX-prueba0.dsic.cloud`
  - `$ rdesktop tsr-XXX-prueba0.dsic.cloud -g80%`

Primeras acciones en nuestra máquina virtual (tsr-XXX-prueba0.dsic.cloud)

- MUY IMPORTANTE: Cambiar nuestro password de **root** utilizando

```
$ passwd
```

- Crear un nuevo usuario (p.e. **pepito**) con su directorio de usuario

```
$ useradd –p /home/pepito –m pepito
```

- Asignar password al nuevo usuario (se recomienda utilizar el usuario y password del dsic para evitar olvidos)

```
$ passwd  pepito
```

- Deshabilitar el cortafuegos para permitir el acceso a nuestros puertos desde una máquina remota (puertos 8000-8100)

```
$ firewall-cmd --permanent --add-port=8000-8100/tcp
$ firewall-cmd --reload
```

- Copiar los archivos de ejemplos de la primera sesión de prácticas desde el directorio **“asigDSIC/ETSINF/tsr/curso1617/ficherosLab1”** del linuxdesktop al **“/home/pepito/lab1”** de vuestra máquina virtual.

***Utilizar la aplicación “conectar con el servidor” descrita en la página 18 del boletín**

netClient.js (pag. 8)

```
var net = require('net');
var client = net.connect({port:8000},
    function() { //connect listener
        console.log('client connected');
        client.write('world!\r\n');
    });
 client.on('data',
    function(data) {
        console.log(data.toString());
        client.end(); //no more data written to the stream
    });
 client.on('end',
    function() {
        console.log('client disconnected');
    });
```

netClient2.js (puede conectar a un servidor remoto)

```
var net = require('net');
var client = net.connect({port:8000, host='localhost'},
    function() { //connect listener
        console.log('client connected');
        client.write('world!\r\n');
    });
 client.on('data',
    function(data) {
        console.log(data.toString());
        client.end(); //no more data written to the stream
    });
 client.on('end',
    function() {
        console.log('client disconnected');
    });
```

Estructura general del **Cliente**

```
var net = require('net');
var client = net.connect({port: XXXX, host: ..... },
    function() { // Listener del evento 'connect'
        console.log('client connected');
        Peticion = ..... //construir la petición
        client.write(Peticion);
    });
client.on('data',
    function(data) { // Listener del evento 'data'
        console.log(data.toString()); // Visualizar la respuesta
        client.end();  // cerrar el socket por parte del servidor
    });
client.on('end',
    function() { // Listener del evento 'end'
        console.log('client disconnected');
    });
```

connect

data

end

Gestiona 3 Eventos

netServer.js (pag. 8)

```
var net = require('net');
var server = net.createServer( function(c) { //connection listener
        console.log('server: client connected');
        c.on('end', function() {
                console.log('server: client disconnected');
        });
        c.on('data', function(data) {
                c.write('Hello\r\n'+ data.toString()); // send resp
                c.end(); // close socket
        });
  });

server.listen(8000, function() { //listening listener
        console.log('server bound');
  });
```

Estructura general del **Servidor**

Gestiona 3 Eventos

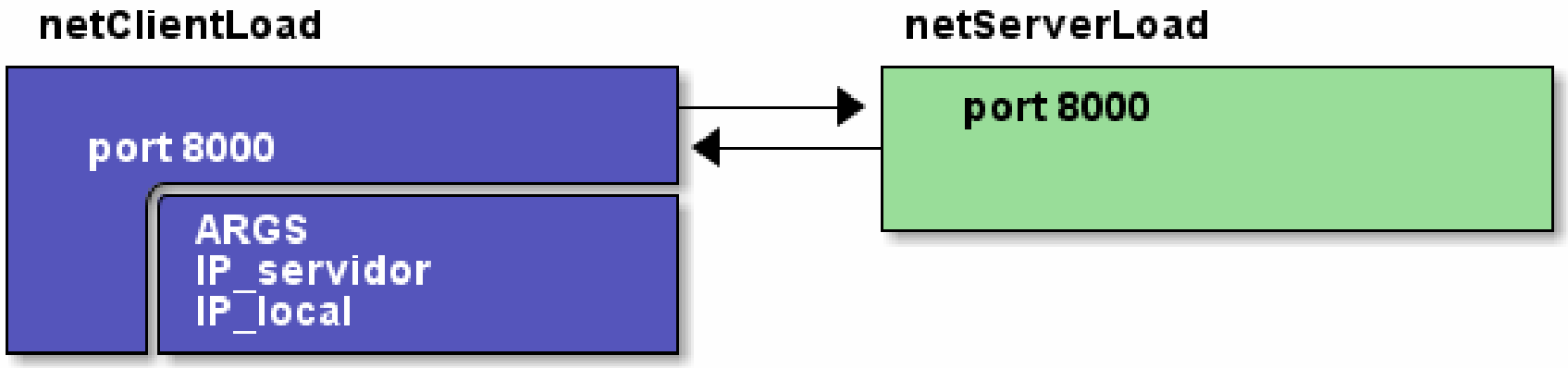
```
var net = require('net');
var server = net.createServer( function(c) {
        console.log('server: client connected');
        c.on('end', function() {
            console.log('server: client disconnected');
        });
        c.on('data', function(data) {
            Respuesta = f(data) // construye respuesta
            c.write(Respuesta); // envia respuesta
            c.end(); // Cierra socket
        });
});
server.listen(8000, function() { //listening listener
        console.log('server bound');
});
```

end

data

connect

2.1 Comprobar la carga de un equipo



- Utilizamos como base **netClient.js** y **netServer.js**
- Ahora el cliente (**netClientLoad.js**) lee dos argumentos por la línea de comandos
- Envía al servidor un mensaje con su dirección IP (**IP_local**)
- El servidor (**netServerLoad.js**) devuelve un mensaje concatenando la dirección IP del cliente con la carga del propio servidor

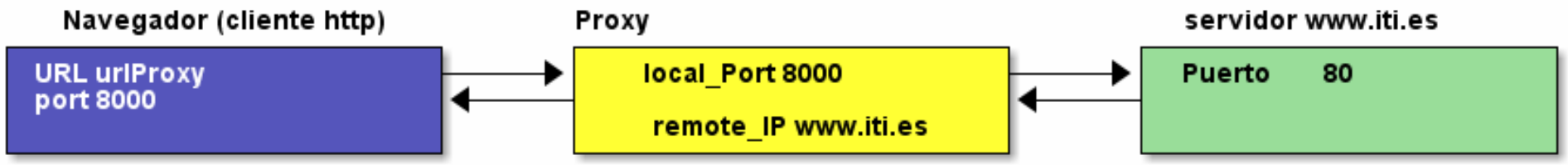
netClientLoad.js

```
var net = require('net');
if (process.argv.length == 4) {
      IP_remota = process.argv[2].toString();
      IP_local = process.argv[3].toString();
   }
else {
     console.log('número de argumentos incorrecto');
     process.exit ();
}

.........

.........
```

netServerLoad.js

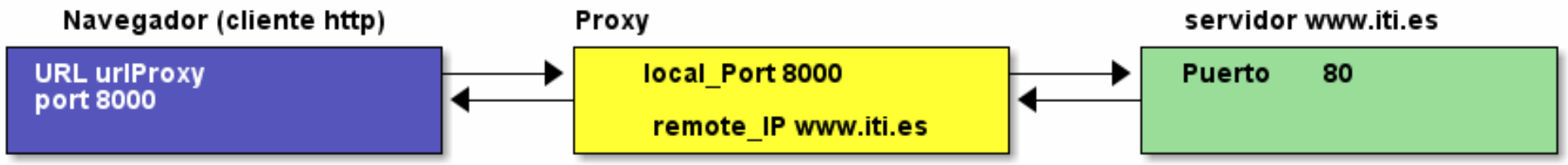
```
var net = require('net');
var fs = require('fs');

function getLoad(){
  data=fs.readFileSync("/proc/loadavg"); //requereix fs
  var tokens = data.toString().split(' ');
  var min1  = parseFloat(tokens[0])+0.01;
  var min5  = parseFloat(tokens[1])+0.01;
  var min15 = parseFloat(tokens[2])+0.01;
   return min1*10+min5*2+min15;
};
.....
.....
```

2.2 Proxy transparente (proxy.js)



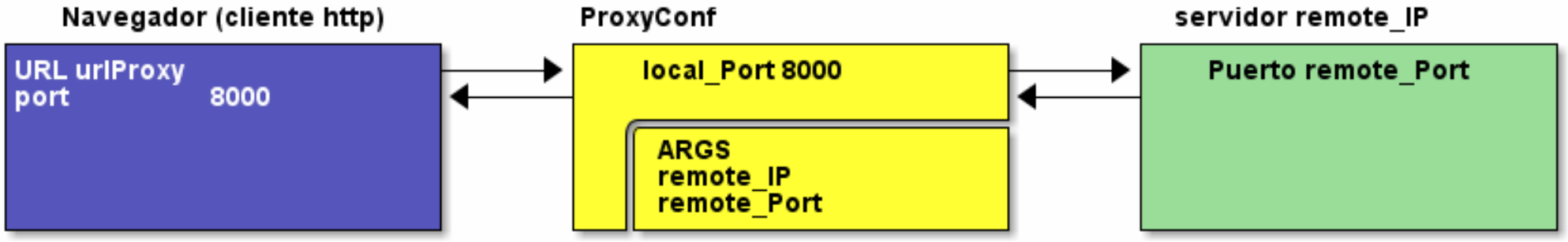

```
var net = require('net');

var LOCAL_PORT  = 8000;
var LOCAL_IP  = '127.0.0.1';
var REMOTE_PORT = 80;
var REMOTE_IP = '158.42.156.2'; // www.iti.es

var server = net.createServer(function (socket) {
        socket.on('data', function (msg) {
            var serviceSocket = new net .Socket();
            serviceSocket.connect(parseInt(REMOTE_PORT), REMOTE_IP, function (){
                serviceSocket.write(msg);
            });
            serviceSocket.on('data', function (data) {
                socket.write(data);
            });
        });
}).listen(LOCAL_PORT);
console.log("TCP server accepting connection on port: " + LOCAL_PORT);
```

2.2 Proxy transparente (proxy2.js)



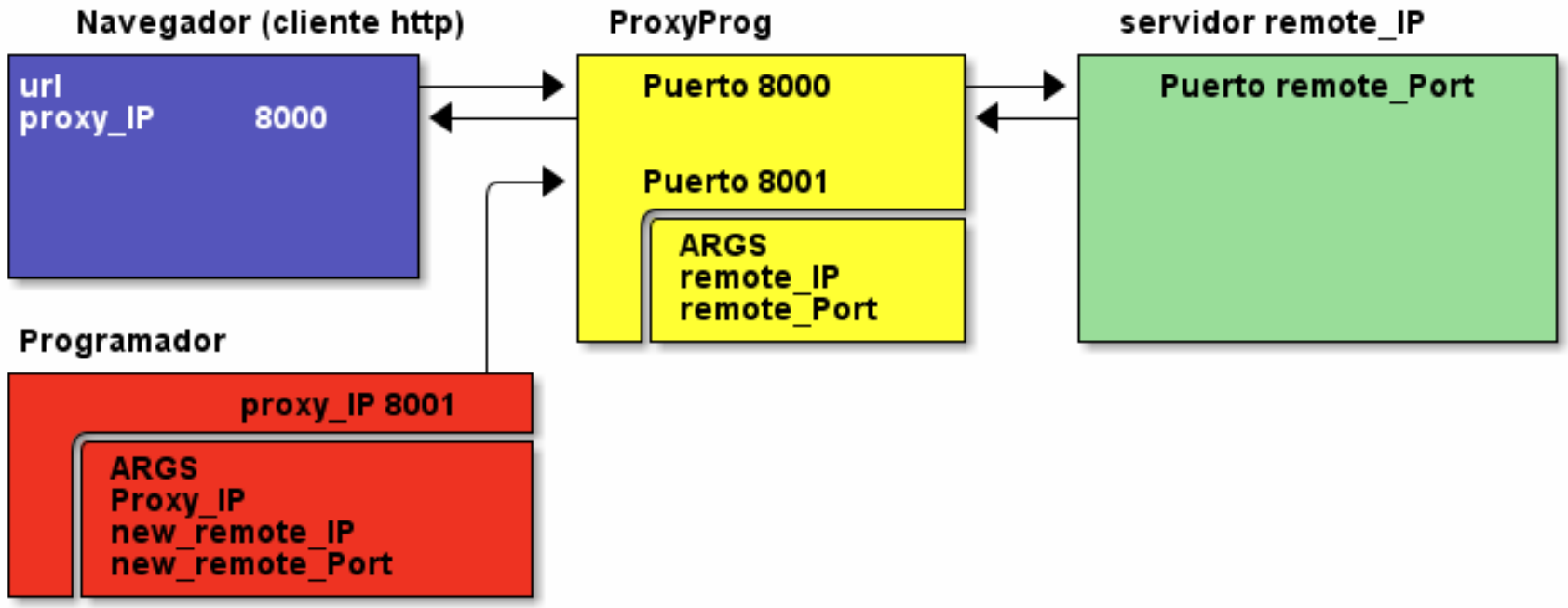

```
var net = require('net');

.....

var server = net.createServer(manejador).listen(LOCAL_PORT);
console.log("TCP server accepting connection on port: " + LOCAL_PORT);

function manejador (socket) {   // el código del proxy es más lejible
        socket.on('data', function (msg) {
            var serviceSocket = new net .Socket();
            serviceSocket.connect(parseInt(REMOTE_PORT), REMOTE_IP, function (){
                serviceSocket.write(msg);
            });
            serviceSocket.on('data', function (data) {
                socket.write(data);
            });
        });
}
```

2.2 Proxy transparente (proxyConf.js)



- Básicamente igual que **proxy.js**
- Se obtiene el endpoint (IP y puerto) del servidor a través de la línea de comandos
- Ahora el proxy puede conectar con cualquier servidor remoto
- Podemos utilizarlo para conectar **netClientLoad.js** con **netServerLoad.js**
- Si trabajamos en local hay que evitar conflictos en el acceso a puertos

2.2 Proxy transparente (proxyProg.js)



- Se puede reprogramar el endpoint del servidor remoto sin detener el proxy
- Para ello necesitamos tener un puerto de programación (**puerto 8001**)
- El nuevo proxy (**proxyProg.js**) atenderá peticiones en este puerto
- Deberemos implementar un programador (**programador.js**) para enviar peticiones al puerto de programación con un nuevo endpoint de servidor

2.2 Proxy transparente (proxyProg.js)

- Para atender peticiones en el puerto 8000 y 8001 el nuevo proxy (**proxyProg.js**) deberá hacer una nueva invocación al método **net.createServer**

- La estructura del nuevo proxy deberá podría ser similar a la siguiente:

```
...
var LOCAL_PORT  = 8000;
var PROG_PORT  = 8001;

...
var server = net.createServer(manejador).listen(LOCAL_PORT);
console.log("TCP server accepting connection on port: " + LOCAL_PORT);

var server2 = net.createServer(manejador2).listen(PROG_PORT);
console.log("TCP server accepting connection on port: " + PROG_PORT);

function manejador (socket) {.....}

function manejador2 (socket) {.....}
```

2.2 Proxy transparente (programador.js)

- **Programador.js** actuará como un cliente que envía peticiones al puerto de programación (puerto 8001) de **proxyConf.js**
- **Programador.js** leerá 3 argumentos por línea de comandos:

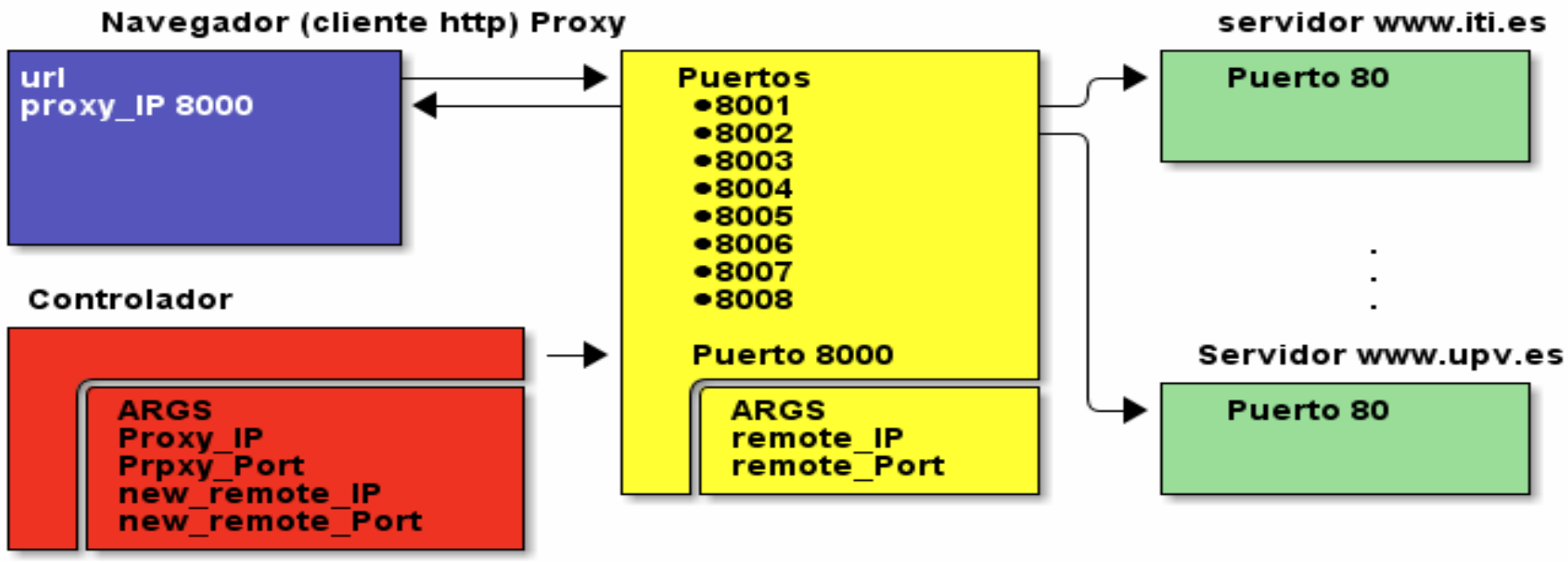
  Proxy_IP : dirección IP del proxy (el puerto siempre es 8001, no se pasa)

  New_remote_IP y **New_remote_Port** : endpoint del nuevo servidor remoto

- **Programador.js** enviará la petición al proxy codificada en formato JSON, mediante el siguiente mensaje:

```
var msg = JSON.stringify ({'remote_ip': New_remote_IP,
'remote_port': New_remote_Port})
```

3 Proxy inverso (proxyInv.js)




```
var msg = JSON.stringify ({'op': 'set', 'inPort': proxy_Port,
'remote': {'ip' :new_remote_IP, 'port' : new_remote_Port}})
```

3 Proxy inverso (proxyInv.js) → Incorrecto!!

```
var net =require('net');

servidoresR ={}

servidoresR[8001]={Port: 80, IP: 'www.upv.es'},
…
…
servidoresR[8008]={Port: 80, IP: 'www.disca.upv.es'}

for (i=8001;i<=8008;i++) {
  var server = net.createServer (manejador(i)).listen(i);
  console.log ('TCP server accepting connection on port ' + i)
}

var server2 =net.createServer (controlador).listen (8000);

function manejador(socket)  {  // La variable i es global a manejador.
    … // el valor de i será 8009 → ERROR
}

function controlador (socket) { … }
```

3 Proxy inverso (proxyInv.js) → Correcto!!

```
var net =require('net');

servidoresR ={}

servidoresR[8001]={Port: 80, IP: 'www.upv.es'},
…
…
servidoresR[8008]={Port: 80, IP: 'www.disca.upv.es'}

for (i=8001;i<=8008;i++) {
  var server = net.createServer (manejador(i)).listen(i);
  console.log ('TCP server accepting connection on port ' + i)
}

var server2 =net.createServer (controlador).listen (8000);


function manejador(i) {    // clausura, para que i sea local dentro de manejador
     return function (socket)  {
          …  // cada manejador lanzado tendrá su propio valor de i

    }
}

function controlador (socket) { … }
```