TSR - PRÀCTICA 1 CURS 2015/16

INTRODUCCIÓ AL LABORATORI I NODE.JS

PROXY INVERS TCP/IP

Aquesta pràctica consta de quatre sessions i té els següents objectius:

1. Introduir els procediments i les eines necessàries per al treball en el laboratori de TSR
    - Identificació i configuració dels recursos existents als laboratoris
    - Aprendre a realitzar una aplicació mínima, amb totes les operacions necessàries que permeten posar-la en funcionament en el laboratori
2. Utilitzar la programació asíncrona per a la gestió de peticions en un servidor (entorn node.js, programació per callbacks, ús de promeses)
3. Desenvolupar una solució per a clients remots basada en el concepte de Proxy Invers

El boletín s'organitza en tres apartats, cadascun dels quals cobreix el corresponent objectiu. Al costat de cada apartat apareix una ordenació temporal orientativa (sessió o sessions en les quals completar aquest apartat):

1. Introducció al laboratori de TSR (programari disponible, procediment de treball). (sessió 1)
2. Primers passos en Node.js (sessions 1 i 2)
3. Desenvolupament d'un proxy TCP/IP invers (sessions 3 i 4)

Clàusula de bon ús

Els recursos que es posen a la disposició de l'alumne es justifiquen pel seu caràcter educatiu, adequat per a la titulació en què s'estan formant. La utilització de recursos tan genèrics com a màquines virtuals, connectats a Internet (amb limitacions) i funcionant ininterrompudament és responsabilitat de l'alumne, qui adquireix el compromís, entre uns altres, de vetlar per la confidencialitat de les seues claus d'accés.

Si entén com ús inadequat **l'intent d'interferir** en l'activitat dels altres companys, accedint a les seues màquines virtuals o afectant al seu treball.

Qualsevol ús que es desvie del propòsit per al qual van ser concebuts aquests recursos serà sancionable. Per a poder fer-ho s'estableixen mecanismes de seguiment en les instal·lacions que monitorizan les informacions necessàries, i que serien aportades com a prova davant l'instructor de la sanció.

CONTINGUT

1 Introducció al laboratori de TSR 3
2 Primers passos en node.js 5
2.1.1 Accés a fitxers 5
2.1.2 JSON (Javascript Object Notation) 9
2.1.3 Programació asíncrona: esdeveniments 9
2.1.4 Interacció client/servidor: mòdul net 10
2.1.5 Encadenament de promeses 11
2.1.6 Consultar la càrrega d'un equip 11
2.1.7 Intermediari entre 2 equips (proxy transparent) 13
3 Apartat 3. Aplicació final: proxy TCP/IP invers 17

1 INTRODUCCIÓ AL LABORATORI DE TSR

Les activitats de laboratori es desenvolupen sobre una distribución basada en LINUX (CentOS 7 de 64 bits)[1].

Pot accederse remotament a un entorn similar al del laboratori mitjançant el sistema virtualitzat del DSIC: **evir** (http://evir.dsic.upv.es/, opció LINUX), però es desaconsella el seu ús perquè **no és compatible** amb l'ús **els ports** en node.js[2].

En **els equips d'escriptori** està instal·lat el programari de desenvolupament necessari

1. Editor de textos: podem utilitzar qualsevol dels instal·lats (ej gedit, geany, etc.). Una bona alternativa (gratuïta i disponible en totes les plataformes) és Visual Code
1. Entorn node.js
2. Biblioteques auxiliars bàsiques (http, net, bluebird, ..)
3. Gestor de paquets npm: permet instal·lar qualsevol altra biblioteca que requerisca la nostra aplicació

Existeixen restriccions en l'ús de ports (només queda lliure per a l'usuari el rang 8000 a 8010), i es treballa amb adreces virtuals → la comunicació entre equips dins del laboratori pot ser complexa.

Node.js està preinstalat en els equips de laboratori. Podem executar:

1. **`node`** per a accedir al shell de node.js: obri indicador ">" i permet escriure codi que s'avalua en prémer return, després de la qual cosa mostra el resultat. Resulta molt útil per a experimentar, depurar codi, etc
   - Podem utilitzar TAB para autocompletar ordre
   - Les tecles up/down permeten desplaçar-se en la història d'ordres
   - La variable especial _ representa el resultat anterior
   - NOTA.- **`var id = expr`** guarda el resultat de l'expressió en **id** però no retorna res, mentre que **`id = expr`** guarda resultat en **id** i retorna aquest resultat
2. **`node fich.js`** per a executar el codi del fitxer fich.js en l'entorn node

El primer pas és un test inicial de la instal·lació. Per a açò anem a escriure el codi d'un servidor web que es limita a saludar al client que contacta amb ell. Els passos a seguir són:

1. Escrivim el sesrvidor web en javascript
   - Escrivim un fitxer amb el siguienti codi (els comentaris en la part dreta no formen part del codi, i només s'inclouen per a entendre-ho millor)

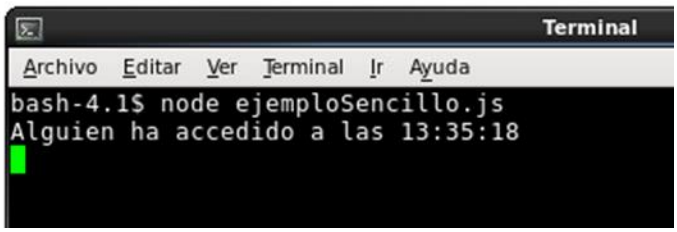
[1] El butlletí es pot desenvolupar en altres entorns (Windows o MacOS) i altres variants de Linux. L'alumne pot trobar la informació necessària en https://nodejs.org/

[2] Ocasionalment las sessions **evir** interfereixen entre si quan els seus processos coincideixen en els ports que desitja emprar l'aplicació

Codi (ejemploSencillo.js)	comentari
`var http = require('http');`	Importa mòdul http
`function dd(i) {return (i<10?"0":"")+i;}`	dd(8) -> "08" dd(16) -> "16"

`var server = http.createServer(`
`  function (req,res) {`
`  res.writeHead(200,{'Content-Type':'text/html'});`
`  res.end('<marquee>Node i Http</marquee>');`
`  var d = new Date ();`
`  console.log('algú ha accedit a les '+`
`  d.getHours() +":"+`
`  dd(d.getMinutes()) +":"+`
`  dd(d.getSeconds()));`
`}).listen(8000);`
	crea el servidor i
	li associa aquesta
	funció
	que retorna una
	resposta fixa
	i
	a més escriu l'hora en
	la consola
	
	El servidor escolta en
	el port 8000

2. Executem el fitxer en l'entorn node.js
3. Utilitzem un navegador web com a client
   - Accedim a la URL **`http://localhost:8000`**
   - Comprovem en el navegador la resposta del servidor, i en la consola el missatge escrit pel servidor



Recomanacions:

1. Evita “copiar i pegar” els programes des del document PDF, ja que podem inserir símbols inadequades que introduïsquen errors sintàctics difícils de depurar. Els programes són curts, per la qual cosa s'aconsella teclejar-los de nou
2. És aconsellable utilitzar un directori específic per a aquesta pràctica, i crear dins del mateix els subdirectoris que es consideren necessaris per a les diferents activitats que planteja el butlletí

2 PRIMERS PASSOS EN NODE.JS

Repassem mitjançant alguns exemples senzills els aspectes bàsics de Javascript i Node.js introduïts en classe. D'aqueixa manera es poden provar empíricament aquestes eines, i posteriormente les utilitzarem per a crear aplicacions pròpies.

2.1.1 Possibles fonts d'error

Durant el desenvolupament de les pràctiques poden aparèixer diferents tipus d'errors. A continuació descrivim els casos comuns i com abordar la solució

- Errors sintàctics. Es detecten en interpretar el codi. S'indica tipus d'error i la seua ubicació en el codi

```
> while 2<3 {console.log("...")}
SyntaxError: Unexpected number
    at Object.exports.createScript (vm.js:44:10)
    at REPLServer.defaultEval (repl.js:117:23)
    at bound (domain.js:254:14)
    at REPLServer.runBound [as eval] (domain.js:267:12)
    at REPLServer.<anonymous> (repl.js:279:12)
    at REPLServer.emit (events.js:107:17)
    at REPLServer.Interface._online (readline.js:214:10)
    at REPLServer.Interface._line (readline.js:553:8)
    at REPLServer.Interface._ttyWrite (readline.js:830:14)
    at ReadStream.onkeypress (readline.js:109:10)
```

- Errors lògics (errors de programació).- Ex. intentar accedir a una propietat de 'undefined', invocar una funció asíncrona sense indicar callback, passar un string quan s'esperava un objecte, intentar accedir a una propietat inexistent, problemes amb dades introduïdes per l'usuari (ex. format de data incorrecte), etc. Han de resoldre's modificant el codi (ex. verificar sempre les restriccions a aplicar sobre els arguments a les funcions).

```
> function suma(array) {return array.reduce(function(x,y){return x+y})}
undefined
> suma([1,2,3])
6
> suma(1)
TypeError: undefined is not a function
   at suma (repl:1:36)
   at repl:1:1
   at REPLServer.defaultEval (repl.js:132:27)
   at bound (domain.js:254:14)
   at REPLServer.runBound [as eval] (domain.js:267:12)
   at REPLServer.<anonymous> (repl.js:279:12)
   at REPLServer.emit (events.js:107:17)
   at REPLServer.Interface._online (readline.js:214:10)
   at REPLServer.Interface._line (readline.js:553:8)
   at REPLServer.Interface._ttyWrite (readline.js:830:14)
```

NOTA.- en tractar-se d'un llenguatge dinàmic sense una orientació a tipus forta, part dels errors que en altres llenguatges (ej Java) captura el compilador, només apareixen ací durant l'execució

- Errors operacionals.- corresponen a problemes detectats en execució a pesar que el codi del programa és correcte (són part del funcionament normal del programa). Poden deure's a l'entorn (ex. ens quedem sense memòria, massa fitxers oberts), configuració del sistema (ex. no hi ha una ruta cap a un host remot), ús de la xarxa (ex. problemes amb l'ús de sockets), problemes d'accés a un servei remot (ex. no puc connectar amb el servidor), ...
  Javascript disposa de construccions try, catch, throw, amb un significat similar al que tenen en Java
  El programador pot dissenyar diverses estratègies de solució en el codi:
  - Quan és clar com resoldre l'error, gestionar directament aquest error (ej davant un error en obrir un fitxer per a escriptura, potser siga adequat crear-ho com un fitxer nou)
  - Quan la gestió de l'error no és responsabilitat d'aqueix fragment de programa, propagar l'error al seu client (a qui ho invoca)
  - Per a errors que poden ser transitoris (ej problemes amb la xarxa), reintentar l'operació
  - Si no podem gestionar ni propagar l'error, i impedeix continuar amb el programa, avortar
  - En un altre cas, simplement anotar l'error (ej en un log)

Per a evitar en la mesura que siga possible els diferents tipus d'errors, es recomana documentar correctament cada funció d'interfície

- els arguments: significat i tipus de cadascun, així com qualsevol restricció addicional
- quin tipus d'errors operacionals poden aparèixer, i com es van a gestionar
- el valor de tornada

i utilitzar el paquet **assert** en el codi (amb les operacions bàsiques **equal**(expr1,expr2, mensajeError) i **ok**(exprLógica, mensajeError)

Exemple

```
/*
 * Make a TCP connection to the given IPv4 address.  Arguments:
 *    ip4addr        a string representing a valid IPv4 address
 *    tcpPort        a positive integer representing a valid TCP port
 *    timeout        a positive integer denoting the number of milliseconds
 *                   to wait for a response from the remote server before
 *                   considering the connection to have failed.
 *    callback       invoked when the connection succeeds or fails.  Upon
 *                   success, callback is invoked as callback(null, socket),
 *                   where `socket` is a Node net.Socket object.  Upon failure,
 *                   callback is invoked as callback(err) instead.
 *
 * This function may fail for several reasons:
```

```
 *    SystemError    For "connection refused" and "host unreachable" and other
 *                   errors returned by the connect(2) system call.  For these
 *                   errors, err.errno will be set to the actual errno symbolic
 *                   name.
 *    TimeoutError   Emitted if "timeout" milliseconds elapse without
 *                   successfully completing the connection.
 *
 * All errors will have the conventional "remoteIp" and "remotePort" properties.
 * After any error, any socket that was created will be closed.
 */
function connect(ip4addr, tcpPort, timeout, callback) {
  assert.equal(typeof (ip4addr), 'string', "argument 'ip4addr' must be a string");
  assert.ok(net.isIPv4(ip4addr), "argument 'ip4addr' must be a valid IPv4 address");
  assert.equal(typeof (tcpPort), 'number', "argument 'tcpPort' must be a number");
  assert.ok(!isNaN(tcpPort) && tcpPort > 0 && tcpPort < 65536,
      "argument 'tcpPort' must be a positive integer between 1 and 65535");
  assert.equal(typeof (timeout), 'number', "argument 'timeout' must be a number");
  assert.ok(!isNaN(timeout) && timeout>0, "argument 'timeout' must be a positive int");
  assert.equal(typeof (callback), 'function');
  /* do work */
}
```

2.1.2 Accés a fitxers

Tots els mètodes corresponents a operacions sobre fitxers apareixen en el mòdul 'fs'. Les operacions són asíncrones, però per a cada funció asíncrona **xx** sol existir la variant síncrona **xxSync**

- Llegir el contingut d'un fitxer

```
var fs = require('fs');
fs.readFile('/etc/hosts', 'utf8', function (err,data) {
  if (err) {
    return console.log(err);
  }
  console.log(data);
});
```

- Escriure contingut en un fitxer

```
var fs = require('fs');
fs.writeFile('/tmp/f', 'contingut del nou fitxer', 'utf8',
  function (err,data) {
  if (err) {
    return console.log(err);
  }
  console.log('s'ha completat l'escriptura');
});
```

- Accés a directoris

  El següent exemple obté tots els fitxers continguts en el directori indicat o en algun dels seus subdirectoris

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

```
function getFiles (dir, files_){
    files_ = files_ || [];
    var files = fs.readdirSync(dir);
    for (var i in files ){
        var name = dir + '/' + files[i];
        if (fs.statSync(name).isDirectory()){
            getFiles(name, files_);
        } else {
            files_.push(name);
        }
    }
    return files_;
}

console.log(getFiles('.')) // directori on buscar
```

- Mòdul path

  Conté diverses funcions que simplifiquen la manipulació de rutes (noms de fitxers o directoris, que poden contenir el separador **/** i els noms especials **.** i **..)**

  - normalize. A partir d'una tira representant el path, interpreta els separadors i els noms especials, i retorna una nova tira que correspon a aqueixa mateixa ruta normalitzada

```
> var path = require('path');
> path.normalize('/a/.///b/d/../c/')
'/a/b/c'
```

  - join. A partir d'una llista variable d'arguments, els uneix i normalitza el path resultant, retornant la tira que correspon al path normalitzat

```
> var path = require('path');
> var url = '/index.html';
> path.join(process.cwd(), 'static', url);
'/home/nico/static/index.html'
```

  - basename, extname i dirname. Permeten extraure els diferents components d'un path

```
> var path = require('path')
> var a = '/a/b/c.html'
> path.basename(a)
'c.html'
> path.extname(a)
'.html'
> path.dirname(a)
'/a/b'
```

  - exists. Permet comprovar l'existència o no d'un path concret

```
> var path = require('path')
> path.exists('/etc', function(exists){
    console.log("Does the file exist?", exists)})
> Does the file exist? true
```

2.1.3 JSON (Javascript Object Notation)

És un format per a representar dades que s'ha convertit en estandard de facto per a la web. Permet representar llistes de valors i objectes (diccionaris amb parells clau/valor):

- Una llista es representa com [a,b,c]
- Un objecte es representa com a {"k 1":v1, "k2":v2, ..}
  - Todes les propietats d'un objecte han d'aparèixer entre ""
  - Els valors d'un objecte poden ser tires, nums, logic, objecte, array o null, però NO funcions, Date, etc
- Las tires poden contenir qualsevol caracter unicode
- Els numeros sempre en base 10

Les operacions fonamentals són:

- JSON.parse(string) construeix un objecte javascript a partir de la seua representació JSON
- JSON.stringify(obj) construeix la representació JSON d'un objecte javascript

```
function PoligonoRegular (lados,longitud) { // constructor de la classe
   this.numLados     = lados;
   this.longitudLado = longitud;
}

PoligonoRegular.prototype.perimetro = function() { // metodo de la classe
   return this.numLados * this.longitudLado;
}

function Cuadrado(longitud) { // constructor de la classe
   this.numLados     = 4;
   this.longitudLado = longitud;
}

Cuadrado.prototype = Object.create(PoligonoRegular.prototype); // herència
Cuadrado.prototype.constructor = Cuadrado;

var c = new Cuadrado (6); // creacion instància

Cuadrado.prototype.superfície = function() { // metodo de la classe
   return this.longitudLado * this.longitudLado;
}

console.log('el perimetro de un cuadrado de lado 6 és '+ c.perimetro());
console.log('y su superficie es '+c.superfície());

console.log(JSON.stringify(c)); // estado del objeto c como string
```

2.1.4 Programació asíncrona: esdeveniments

single.js

```
function fib(n) {return (n<2)? 1: fib(n-2)+fib(n-1);}

console.log("iniciant ejecucion...");

setTimeout( //espera 10 seg i executa la funcion
   function() {console.log('M1: Vull escriure açò...');},
   10);
```

	`var j = fib(40); // aquesta invocacion costa mes de 1seg`
	
	`function otherMsg(m,o) {console.log(m + ": El resultat és "+o);}`
	
	`otherMsg("M2",j); //M2 s'escriu abans que M1 perquè el fil principal rares vegades se suspèn`
	
	`setTimeout( // M3 s'escriu després de M1`
	`  function() {otherMsg('M3',j);},`
	`  1);`

eventsimple.js	`var ev = require('events');`
	`var emitter = new ev .EventEmitter;`
	`var i1 = "print", i2= "read";  // name of events`
	`var num1 = 0, num2 = 0;  // auxiliary vars`
	
	`// register listener functions on the event emitter`
	`emitter.on(i1,`
	`  function() {console.log('event '+i1+' has happened '+num1+' estafes')})`
	`emitter.on(i2,`
	`  function() {console.log('event '+i2+' has happened '+num2+' estafes')})`
	`emitter.on(i1, // habite than one listener for the same event is possible`
	`  function() {console.log('something has been printed!')})`
	
	`// generate the events periodically`
	`setInterval(`
	`  function() {emitter.emit(i1);}, // generates i1`
	`  2000);  // every 2 seconds`
	`setInterval(`
	`  function() {emitter.emit(i2);}, // generates i2`
	`  8000);  // every 8 seconds`

2.1.5 Interacció client/servidor: mòdul net

Client (netclient.js)	`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');`
	`  });`

Servidor (netserver.js)	`var net = require('net');`

```
var server = net.createServer(
   function(c) { //connection listener
      console.log(client connected');
      c.on('end',
         function() {
            console.log(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');
   });
```

2.1.6 Encadenament de promeses

```
var promise = require('bluebird'),
    fs      = require('fs');

var readFileSync = promise.promisify(fs.readFile); //synchronous variant
var f = readFileSync('netServer.js','utf-8');

function printFile (data) {console.log(data);        return data;} // aux function
function showLength(data) {console.log(data.length); return data;} // aux function
function showErrors(err)  {console.log('Error reading file ..'+err);}

f.then(printFile).then(showLength,showErrors);
```

2.1.7 Accés a arguments en línia d'ordres

El shell arreplega tots els arguments en línia d'ordres i els hi passa a l'aplicació javascript empaquetats en un vector denominat **process.argv** (abreviatura de' argument values').

Defineix un fitxer args.js amb el següent codi

```
console.log(process.argv);
```

I invoca-ho amb diferents arguments. Per exemple:

```
node args.js un dos tres quatre
```

Pots observar que els dos primers elements del vector corresponen a 'node' i el path del programa executat, respectivament. És freqüent aplicar process.argv.slice(2) per a descartar aquests elements del vector (de manera que només queden els arguments reals per a la nostra aplicació).

2.1.8 Consultar la càrrega d'un equip

A partir de **`netClient`** i **`netServer`** creamos un parell d'aplicacions **`netClientLoad`** i **`netServerLoad`** amb les següents característiques:

- Empren el port 8000
- **`netClientLoad`** aporta la seua IP com a cos del missatge. Aquesta IP haurà de ser introduïda com a paràmetre en invocar el programa
  - Cal assegurar-se que el procés finalitza (p. ex. amb **`process.exit()`**)
  - En els equips del laboratori, però no en EVIR, es pot esbrinar d'una *forma indirecta* (preguntar-li qui som a algú "de fora"). Ho teniu en el directori de l'assignatura com **`averiguar_ip.sh`**

```
wget -qO - http://memex.dsic.upv.es/cgi-bin/ip
```

- **`netServerLloeu`** contesta amb la seua IP i un nombre que representa la seua càrrega de treball actual, calculada a partir de **`/ proc/loadavg`** de la següent forma:

```
var fs = require('fs');
function getLoad(){
  data=fs.readFileSync("/proc/loadavg");
  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;
};
```

Dóna un pes 10 a la càrrega de l'últim minut, un pes 2 a la dels últims 5 minuts, i un pes 1 a la dels últims 15. A cada ítem li suma una centésima per a evitar la confusió entre el valor 0 i un error; al cap i a la fi, quan ho apliquem, l'interessant serà el valor relatiu, no l'absolut.

Per tant, **`netServerLoad`** no requereix paràmetres d'entrada, i **`netClientLoad`** tindrà dos arguments d'entrada que es passaran a través de la línia d'ordres:

1. Adreça IP remota (Adreça IP del servidor).
2. Adreça IP local (Adreça IP des d'on s'executa el client).

Completa tots dos programes, col·loca'ls en equips diferents i fes que es comuniquen mitjançant el port 8000, de manera que **`netServerLoad`** emeta un missatge per cada petició rebuda amb IP del client, i **`netClientLoad`** mostre la càrrega retornada com a resultat de l'última petició.

Recorda que per a obtenir els arguments de la línia d'ordres en un programa Javascript pots utilitzar les següents instruccions:

- **`process.argv.length`** : permet obtenir el nombre d'arguments passats per la línia d'ordres, incloent el nom del programa.

- **`process.argv[2]`** : permet obtenir el primer argument d'entrada si hem invocat mitjançant "**`node programa arg1 …`**". **`process.argv[0]`** conté la cadena '**`node`**' i **`process.argv[1]`** conté la cadena '**`programa`**'.
    - Molt important el mòdul net per a treballar amb sockets

```
var net = require('net');
  net.createServer(function (socket) {
  console.log('socket connected!');
  // etcètera ...
}).listen(8000)
```

    - A manera de referència, s'ha deixat una instància en marxa en el port 9000 de **`tsr1.dsic.upv.es`**

2.1.9 Intermediari entre 2 equips (proxy transparent)

Volem construir un servei que actua com a intermediari: en ser invocat retorna com a resultat el que es produiria en invocar a un altre, que pot residir fins i tot en un altre equip. És la funcionalitat que ofereix un redirector, com un proxy HTTP, una passarel·la ssh o un servei similar. Més informació en **`http://en.wikipedia.org/wiki/proxy_server`**

En aquest sistema intervenen 4 components:

1. L'invocant del servei, que actua com a client i que emet les peticions. Podem utilitzar un **invocant extern** específic (ex. un navegador web que actua com a client HTTP).
1. L'intermediari , que rep peticions del client i les reencamina al destinatari final.
2. El destinatari final, que implementa realment el **servei**. Li és indiferent que les peticions procedisquen directament del primer actor (invocant) o del segon (intermediari). Podem utilitzar un **servei final extern**, com un servidor de web convencional (ens limitem a protocol HTTP)
2. El **protocol**, que funciona com a especificació dels missatges entre invocant i servei, tant en la seua forma (sintaxi) com en el seu significat (semàntica).

L'intermediari no altera el contingut del missatge més enllà de les propietats aplicables a TCP/IP (és a dir, pot modificar ports i adreces). D'altra banda actua com un transport *neutre* que desconeix el protocol establert entre invocant i servei.

- En resum, mentre es tracte d'un protocol basat en TCP/IP, l'intermediari ha de funcionar

Com a primer experiment vam crear un intermediari que, emprat per un client HTTP, retorne els documents d'un servidor de web preestablit. El programa **`myTcpProxy.js`** actua com un servidor basat en el mòdul net que:

1. Rep peticions en el port 8000
2. Toma la petició i construeix una altra canviant destinació i port
    - Com a servei a invocar, tomaremos el port 80 del servidor web de l'Institut Tecnològic d'Informática de la UPV (ITI) (IP 158.42.156.2)
3. Retorna la resposta del servidor a l'invocante.

Aquest proxy "transparent" és *visible* per al client (al cap i a la fi necessita contactar amb ell). Se comporta com a servidor per a rebre peticions del seu client, i com a client per a enviar les peticions al servidor web real.

Codi (mytcpproxy.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, LOCAL_IP);
console.log("TCP server accepting connection on port : " + LOCAL_PORT);
```

Empra per a açò dos sockets (socket per a comunicar amb el client i serviceSocket per a dialogar amb el servidor):

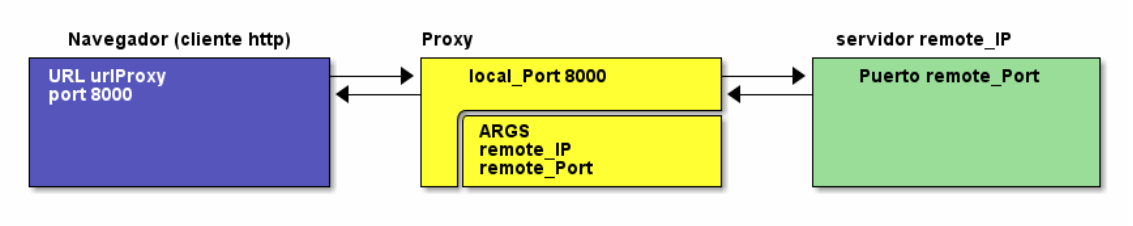
1. llig un missatge (`msg`) del client
2. obri una connexió amb el servidor
3. escriu una còpia del missatge
4. espera la resposta del servidor
5. i retorna una còpia al client

Comprova qué el funcionament és correcte utilitzant un navegador web que apunte a http://direccion_de el_proxy:8000/, i recorda que es pot obtenir la IP del proxy amb **`averiguar_ip.sh`**

NOTA: És possible que en executar un programa en node aparega l'error "EADDRINUSE". Normalment es deu al fet que estem referenciant un port que està sent utilitzat per un altre programa node (ex. perquè algun programa node anterior ha quedat en execució fora del nostre control). Açò es pot determinar executant el comandament '**`ps -all`**' en la línia d'ordenes de Linux. En cas que apareguen programes node en execució podem finalitzar-los

utilitzant el comandament '**`pkill -f node`**'. Aquest comandament finalitza tots els programes node que s'estan executendo amb el nostre identificador d'usuari.

Modifica el codi del proxy perquè reba com a paràmetres d'entrada l'adreça del servidor remot (adreça IP i port). D'aquesta forma **`myTcpProxy`** podrà ser usat d'una forma més flexible.

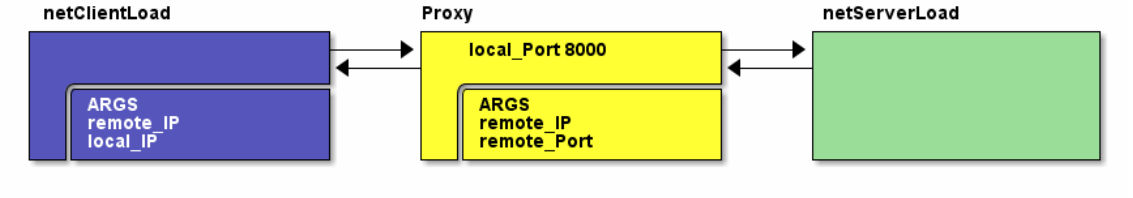


Ara **`myTcpProxy`** tindrà els següents **arguments d'entrada** (en aqueix ordre):

1. Adreça IP remota (Adreça IP del servidor remot).
2. Port remot (Port de recepció de peticions del servidor remot).

Utilitza aquesta nova versió de `myTcpProxy` per a accedir de nou al servidor de**l ITI**, com en l'exemple anterior. Has de seguir utilitzant com a port local d'atenció de peticions el port 8000.

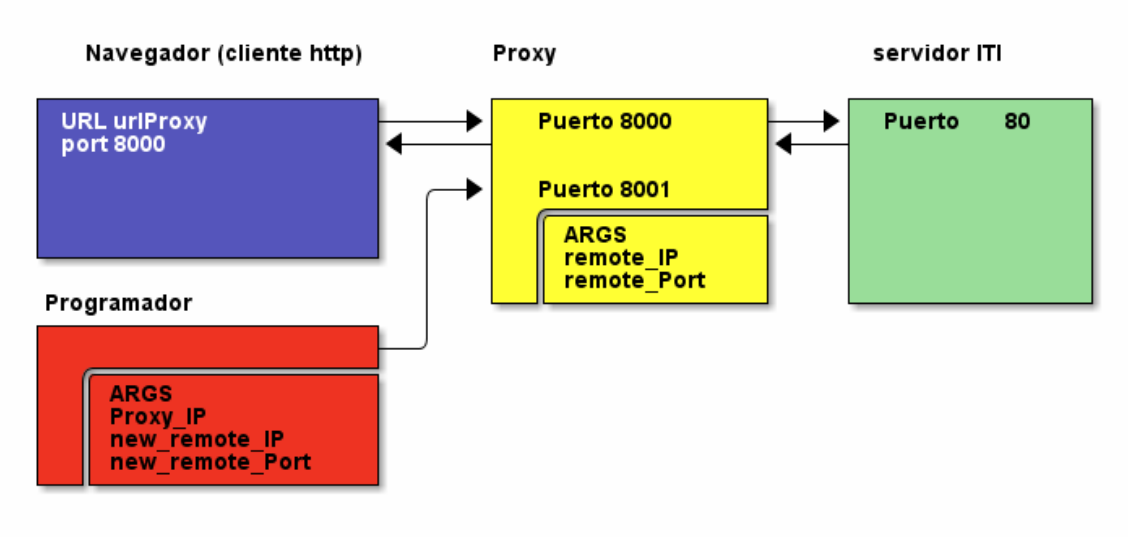
Ara **`myTcpProxy`** ens permet accedir a qualsevol servidor remot. **Utilítzal, invocant-ho amb els paràmetres adequats,** de manera que **`netClientLoad`** i **`netServerLoad`** puguen comunicar-se mitjançant aquest proxy.



En aquest cas **`netClientLoad`** rebrà com a segon paràmetre l'adreça IP del proxy en compte de l'adreça IP de **`netServerLoad`**.

Nota: Si el proxy s'executa en la mateixa màquina que **`netServerLoad`** és necessari canviar el port d'atenció del servidor de càrrega a un diferent de 8000, ja que aquest és el port en el qual atén el proxy).

Crea una variant del proxy cridada `newtcpproxy.js` que utilitze un **port de programació** al que es puguen enviar un parell de valors (IP, port) per a reprogramar el servidor remot.



En aquest cas:

- Les peticiones al port 8001 (port de programació del proxy) enviarien el valor del nou servidor i port remot.
- Les peticions al port 8000 (port d'atenció de peticions del proxy) s'atenen amb la redirecció vigent del servidor remot.

`newTcpProxy` tindrà els mateixos **arguments d'entrada** que **`myTcpProxy`**, amb la finalitat de proporcionar l'adreça d el servidor remot inicial.

1. Adreça IP remota (Adreça IP del servidor remot inicial).
2. Port remot (Port de recepció de peticions del servidor remot inicial).

Nota: el port d'atenció de peticions del proxy segueix sent 8000, i el port de programació serà el 8001. Aquests dos valors no es passen com a paràmetres d'entrada del proxy, ja que el seu valor és fixe!!.

Necessitaràs un programa client (**`programador.js`**) que envie aquests valors (*programe* el proxy). Aquest programa tindrà els següents **arguments d'entrada**:

1. Adreça IP del proxy (adreça IP del proxy).
2. Adreça IP remota (adreça IP del nou servidor remot).
3. Port remot (port de recepció de peticions del nou servidor remot).

Nota: No es passa com a argument el port de programación del proxy, ja que ést'ha de ser sempre 8001. La dirección IP del proxy sí que es passa ja que ens permet utilitzar el programador des d'una màquina no local al proxy.

El **`programador`** deu enviar missatges amb un contingut com el següent:

```
var msg = JSON.stringify ({'remote_ip':"158.42.4.23", 'remote_port':80})
```

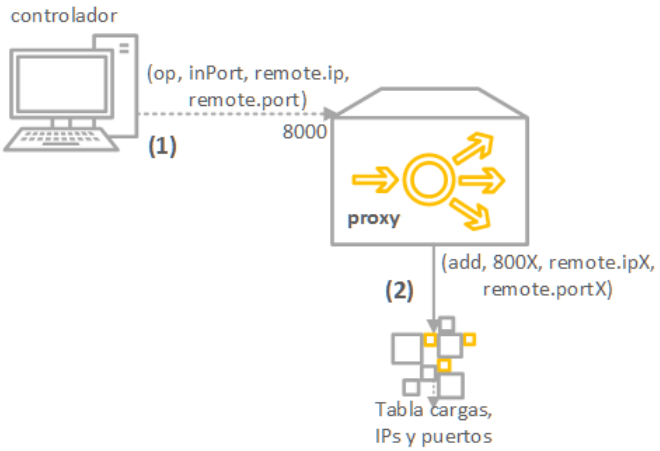
Per a verificar el funcionament de **`newTcpProxy`** el pots provar amb el parell **`netServerLoad`**/**`netClientLoad`**, o bé amb un client HTTP si la destinació és un servidor de web.

3 APARTAT 3. APLICACIÓ FINAL: PROXY TCP/IP INVERS

Un proxy invers és un servidor que col·loquem entre els clients remots i els nostres servidors web:

- Accepta les peticions dels clients
  - Oculta la topoligía dels servidors
  - Pot realitzar autenticació, control d'accés, xifrat, etc.
- Redirigeix el tràfic cap al servidor que corresponga
  - Permet equilibrar la càrrega dels servidors
  - Pot tolerar la fallada d'un servidor
- Permet inspeccionar, transformar i enrutar el tràfic HTTP
  - Auditoria, logs, etc.

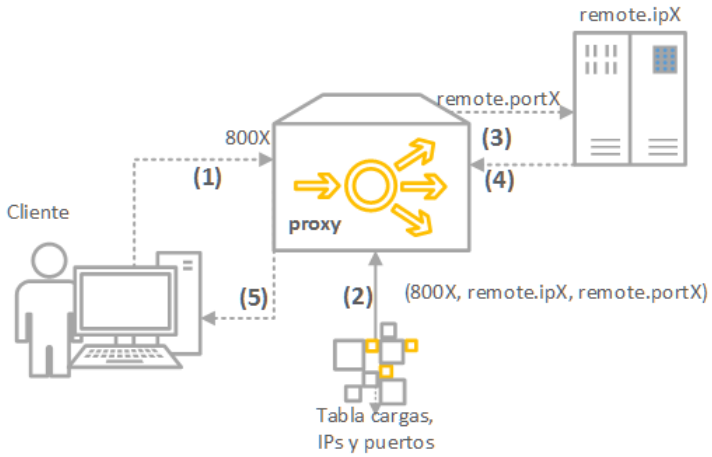
Disposem d'un nombre arbitrari de clients que envien sol·licituds de servei als ports 8001 a 8008[3] del nostre proxy.



Mitjançant el port 8000 (programa **`controlador`**), podem programar al proxy per a indicar-li quin servei ve representat per un port en concret. P. ex., si el proxy rep en el seu port 8000 aquest missatge:

```
var msg =JSON.stringify ({'op':"set", 'inPort':8001, 'remote':{'ip':"158.42.4.23", 'port':80}})
```

… les peticions que arriben des d'aqueix moment al port 8001 del nostre proxy se serviran connectant amb el port 80 de 158.42.4.23. Açò seria equivalent al proxy vist anteriorment (**`mytcpproxy.js`**).

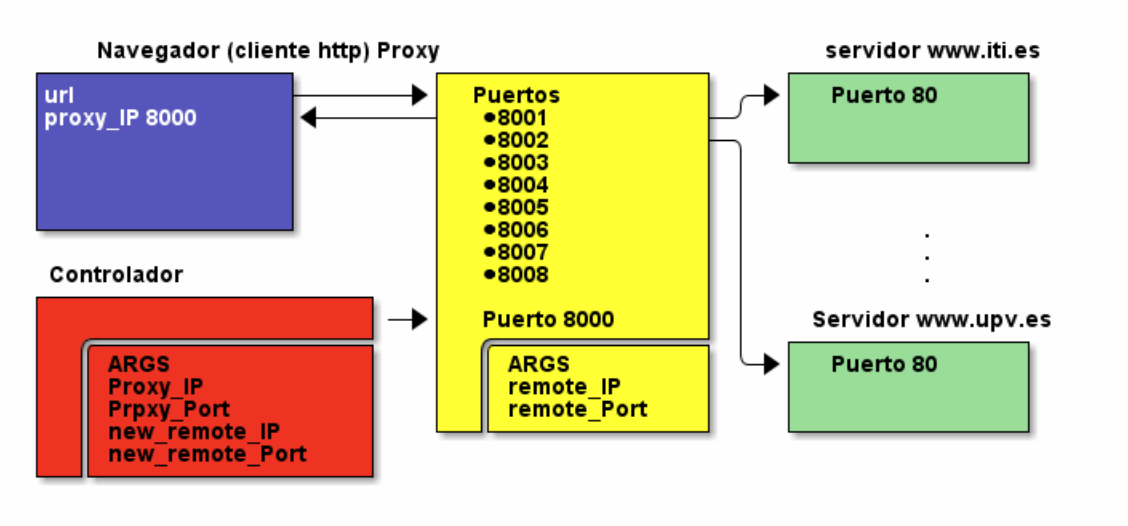


Ha de desenvolupar-se un **programa capaç de comunicar-se amb el proxy per a programar-ho** a través del port de control, i un proxy que implemente la funcionalitat descrita.

- El programa es denominarà **`controlador.js`**, aquest proxy es dirà **`proxy.js`**

[3] Recordem que en el laboratori podem disposar dels ports 8000 a 8010 per als nostres programes

`proxy` no tindrà arguments d'entrada, ja que els ports d'atenció de peticions sempre seran els ports de 8001 a 8008, el port de programació siempre serà el 8000, i les direcciones dels servidors remots inicials són valors constants que s'escriuen dins del programa. Las pruebas a verificar es basaran en l us protocols HTTP, HTTPS i FTP mitjançant un navegador de web.



Seguidament es mostren algunos valors de prova d'aquests servidors remots, a títol d'exemple:

Port del proxy	IP del servei remot	Port del servei remot
8001	www.dsic.upv.es	80 (http)
8002	www.upv.es	80 (http)
8003	www.google.es	80 (https)
8004	www.iti.es	80 (http)
8005	www.disca.upv.es	80 (http)

Però ha d'estar preparat per a atendre qualsevol associació vàlida.

Per la seua banda, **`controlador`** ha de tenir els següents **arguments d'entrada**:

1. Adreça IP del proxy (adreça IP del proxy).
2. Port d'entrada del proxy (Port d'entrada que serà reprogramat).
3. Nova adreça IP remota.
4. Nou port remot.

Nota: els arguments 3 i 4 determinen la nova redirecció de l'entrada al proxy especificada en el segon argument. Els valors vàlids per a aquest segón argument están en el rang 8001 a 8008. El port de programació no es passa com a argument, suposem que sempre és 8000.