



## Pràctica 2

XARXES

GRAU EN ENGINYERIA INFORMÀTICA

*David Pérez Sánchez, 41582033B, david.psnk@gmail.com*  
*Laura Galera Alfaro, 20875366Z, laura.galeraaa@gmail.com*

*Grup de divendres a les 15:00h*

Professor de pràctiques

Pere Vilà

8 de gener de 2021

# Índex

<b>1</b>	<b>Requeriments mínims i millores</b>	<b>3</b>
1.1	Millores . . . . .	3
<b>2</b>	<b>Manual d'usuari</b>	<b>4</b>
2.1	Node . . . . .	4
2.2	Agent . . . . .	6
<b>3</b>	<b>Disseny de l'aplicació</b>	<b>7</b>
3.1	L'arquitectura en capes . . . . .	7
3.2	Diagrama funcional . . . . .	9
3.2.1	Agent . . . . .	9
3.2.2	Node . . . . .	10
<b>4</b>	<b>La interfície aplicació-usuari</b>	<b>11</b>
<b>5</b>	<b>Les capes d'aplicació i transport (Serveis)</b>	<b>12</b>
5.1	Serveis de la capa d'aplicació de MI . . . . .	12
5.2	Serveis de la capa d'aplicació de LUMI . . . . .	12
5.3	Serveis de la capa d'aplicació de DNS . . . . .	12
5.4	Serveis de la capa de transport TCP . . . . .	12
5.5	Serveis de la capa de transport UDP . . . . .	13
<b>6</b>	<b>Les capes d'aplicació i transport (Interfícies)</b>	<b>14</b>
6.1	Interfície de la capa d'aplicació de MI . . . . .	14
6.2	Interfície de la capa d'aplicació de LUMI . . . . .	15
6.3	Interfície de la capa d'aplicació de DNS (part client) . . . . .	16
6.4	Interfície de la capa de transport TCP . . . . .	16
6.5	Interfície de la capa de transport UDP . . . . .	17
6.6	Part compartida de TCP i UDP . . . . .	18
6.7	Part compartida de MI i LUMI . . . . .	19
<b>7</b>	<b>Les capes d'aplicació i transport (Protocols)</b>	<b>20</b>
7.1	Protocol de les capes d'aplicació de MI (PMI) i de LUMI (PLUMI) . . . . .	20
7.2	Protocol de les capes de transport TCP i UDP. . . . .	23
7.3	Estudi amb “ss” . . . . .	25

8 Problemes i suggeriments	28
9 Treball en parella i dedicació	29

# 1 Requeriments mínims i millores

## Requeriments mínims

En l'aplicació que hem desenvolupat, s'han complert els requeriments mínims demanats. A continuació se'n fa un resum:

- Construir una aplicació MI que segueixi el model P2P i faci servir adreces de MI.
- Integrar un agent d'usuari LUMI, que permeti les operacions registre, desregistre i localització.
- Construir un node LUMI d'un determinat domini.
- Llegir un fitxer anomenat "nodelumi.cfg" amb tots els usuaris donats d'alta en el domini.
- Crear una interfície aplicació-usuari que permeti a l'usuari introduir el *nickname*, la seva @MI, per tal de registrar-se, i establir una conversa amb un altre usuari, tant del seu com d'un altre domini, entrant la @MI del company. També pot esperar que algú es posi en contacte amb l'usuari local per iniciar la conversa. Una vegada acabat el xat, s'ha de desregistrar l'usuari i finalitzar l'execució.
- L'aplicació MI i el node LUMI ha de funcionar amb els d'altres estudiants.
- Implementar el protocol PLUMI, que ha d'anar sobre UDP, i el protocol PMI, que va sobre TCP.
- Un agent LUMI ha de parlar només amb el seu node LUMI.
- Fer servir el llenguatge C.
- Estructurar el codi en fitxers, separant la interfície d'aplicació-usuari, les capes d'aplicació MI, LUMI i DNS i les capes de transport TCP i UDP.
- Dissenyar el node LUMI com un servidor iteratiu.
- Escriure en un fitxer de "log" una línia explicativa de cada missatge rebut o enviat per un agent d'usuari o un node LUMI. De la forma: *E: 84.88.154.3:UDP:4500, PRmaria@bb.com, 14*
- Fer el treball en parelles de dos i entregar aquest document.

## 1.1 Millores

També hem implementat algunes de les millores proposades. En concret, hem optat per afegir a la nostra aplicació les tres primeres millores que ens van comentar:

- Una interfície aplicació-usuari de l'aplicació de MI que indiqui a l'usuari per quina raó no s'ha pogut realitzar el registre, el desregistre o una localització

#### **Com s'ha fet:**

Per fer aquesta millora hem modificat les tres funcions del fitxer *ALUMIc.c*; *LUMIc\_DemanaReg()*, *LUMIc\_DemanaDesreg()* i *LUMIc\_DemanaLoc*, per a que retornin un nombre enter indicant si ha anat bé i, en cas que no, la raó per la qual no ha funcionat. Per exemple, en el registre, si l'usuari no existeix a la taula es retorna un 1 per printar per pantalla un missatge informatiu del perquè no s'ha fet el registre. A més, també hem modificat la interfície en mode text per a que sigui visualment més atractiva.

- Una interfície aplicació-usuari del node de LUMI que faciliti a l'administrador la seva gestió.

#### **Com s'ha fet:**

S'ha modificat el codi de nodelumi. El primer pas ha estat crear un *do while* que finalitzi en cas que l'administrador entri un */exit*. Llavors, dins del bucle, es fa una crida a *AHaArribatAlgunaCosa()*, que tan pot ser per teclat com pel *socket* lumi. En cas que sigui per teclat, s'ha de tractar el tipus de comanda, ja sigui */alta* per incloure un nou usuari al domini, */baixa* per treure un usuari donat d'alta, */llistar* per llistar els usuaris del domini, */help* per mostrar el menu o */exit* per acabar el programa. Si és rep un paquet pel *socket* lumi, el funcionament és el mateix que sense la millora. A més, quan es tanca el socket, s'aprofita per copiar tots els valors de la taula al fitxer de configuració.

- El desregistre “anormal” d'un usuari.

#### **Com s'ha fet:**

En primer lloc, a *ALUMIs* hem hagut d'afegir un nou camp a la taula d'usuaris: *int intents*, un per cada usuari donat d'alta. D'aquesta manera, cada vegada que el servidor rep una petició de localització per a l'usuari X, que figura *online*, s'accedeix al seu camp *intents* i s'incrementa en una unitat. Com que nosaltres hem permès que hi hagi un total de 5 retransmissions quan expira el *timeout*, el límit d'intents és 5. Quan es rep la cinquena PL es comprova que *intents* sigui 5 i la petició de localització ja no es reenvia al client, sinó que es posa l'usuari *offline* i es retorna un paquet de resposta localització amb el camp tipus a 3, és a dir, *offline*. A part d'això, també hem hagut d'inicialitzar el comptador a 0 quan es fa un registre i tornar-lo a posar a 0 si es rep una RL procedent del client, ja que resulta que no està *offline*.

## **2 Manual d'usuari**

### **2.1 Node**

Quan s'executa l'aplicació, apareix per pantalla un missatge de benvinguda que mostra el nom del domini del node. També s'indica la comanda a utilitzar perquè es mostri l'ajuda del programa.

```

-----
NODE LUMI INICIALITZAT CORRECTAMENT
DOMINI: david-nuclear
-----
Escriu /help per mostrar l'ajuda
-----
> █

```

Si s'introdueix la comanda `/help`, apareixen totes les funcionalitats disponibles.

```

> /help
Escull una opció:
Escriu /exit per acabar
Escriu /alta nomUsuari per afegir un usuari
Escriu /baixa nomUsuari per eliminar un usuari
Escriu /l·listar per l·listar l'estat i localització dels usuaris
Escriu /help per mostrar l'ajuda
-----
> █

```

Comandes disponibles:

- `/exit` → Acaba l'execució del node.

```

> /exit
-----
> S'ha finalitzat correctament

```

- `/alta nomUsuari` → Dona d'alta l'usuari *nomUsuari* al node. S'ha d'indicar el nom del nou usuari després del nom de la comanda (separat amb un espai). El nou usuari estarà *offline* per defecte. En cas que *nomUsuari* ja estigués prèviament donat d'alta, dona error.

```

> /alta PerePi
Usuari PerePi afegit
-----
> █

```

- `/baixa nomUsuari` → Dona de baixa a l'usuari *nomUsuari* del node. El nom de l'usuari que es vol donar de baixa ha d'anar després del nom de la comanda (separat amb un espai). En cas que *nomUsuari* no estigués prèviament donat d'alta, dona error.

```

> /baixa PerePi
Usuari PerePi tret
-----
> █

```

- `/l·listar` → Mostra per pantalla la informació de tots els usuaris donats d'alta en tres camps: nom, adreça del *socket* LUMI del seu agent i l'estat en què es troba.

```

> /l·listar
FORMAT:  NOM    ->  SOCKET LUMI    ->  ESTAT
david.perez -> No registrat -> OFFLINE
laura.galera -> No registrat -> OFFLINE
pere.vila   -> No registrat -> OFFLINE
PerePi      -> 127.0.0.1:36894 -> ONLINE
-----
> █

```

- `/help` → Mostra el missatge d'ajuda per pantalla.

## 2.2 Agent

Tan bon punt s'executa l'aplicació, es mostra un missatge de benvinguda per pantalla que indica que s'està utilitzant un client MI. També demana a l'usuari que introdueixi el seu *nickname*, que no pot ser més llarg de 20 caràcters.

Un cop s'ha entrat el *nick*, es demana també l'adreça MI de l'usuari. Aquesta ha de tenir el format `nomMI@domini` i no pot ser més llarga de 36 caràcters.

```

-----
Client d'aplicació MI
-----
LOGIN
Introdueix el teu nick (màxim 20 caràcters):
> PerePi
Entra @MI (mida MAX 36 caràcters) nomMI@domini:
> PerePi@domini█

```

A continuació el client procedirà a registrar-se al node i, en cas que tot vagi correctament, es mostrarà per pantalla el següent:


```

S'ha registrat correctament al node
-----
Prem ENTER per crear una connexió o ESPERA a una petició de connexió...
Entra EXIT per tancar el programa
█

```

Aleshores l'usuari pot fer tres coses:

### Prémer ENTER

Si es prem la tecla , l'aplicació permetrà a l'usuari establir una connexió amb un altre usuari. Es demanarà per pantalla l'adreça MI del remot. Aquesta ha de tenir el format `nomMI@domini` i no pot ser més llarga de 36 caràcters. Si l'adreça existeix i està registrada en el domini a la qual pertany, s'iniciarà una conversa amb l'usuari remot, indicat l'èxit en la connexió, els *nicks* i les adreces MI.

```

Entra @MI destí (mida MAX 36 caràcters) nomMI@domini:
> Maria@domini
S'ha pogut localitzar al company de conversa
-----
Nickname Local -> PerePi
Nickname Remot -> Maria

ADREÇA LOCAL -> 192.168.1.53:50603
ADREÇA REMOTA -> 192.168.1.53:37247
-----

```

## ESPERAR

Si l'usuari vol que un altre usuari remot es connecti a ell, ha de restar a l'espera. Quan això succeeixi s'iniciarà una conversa amb l'usuari remot i se li mostrarà per pantalla els *nicks* i les adreces MI.

```

-----
Nickname Local -> Maria
Nickname Remot -> PerePi

ADREÇA LOCAL -> 192.168.1.53:37247
ADREÇA REMOTA -> 192.168.1.53:50603
-----

```

## Entrar EXIT

En cas que es vulgui sortir de l'aplicació, l'usuari pot introduir EXIT. Això farà que es desregistri del domini i que finalitzi l'execució de la seva aplicació.

```

EXIT

T'has desregistrat del teu domini
Fi execució

```

A continuació dels dos primers casos, tant l'usuari local com el remot, podran començar a intercanviar missatges. Tan bon punt un dels dos vulgui finalitzar la conversa i tancar la connexió amb el seu company, haurà d'introduir la comanda `/exit`. Aleshores l'aplicació tornarà a esperar a que algun usuari remot s'hi connecti o que l'usuari local realitzi alguna acció de les indicades per pantalla.

```

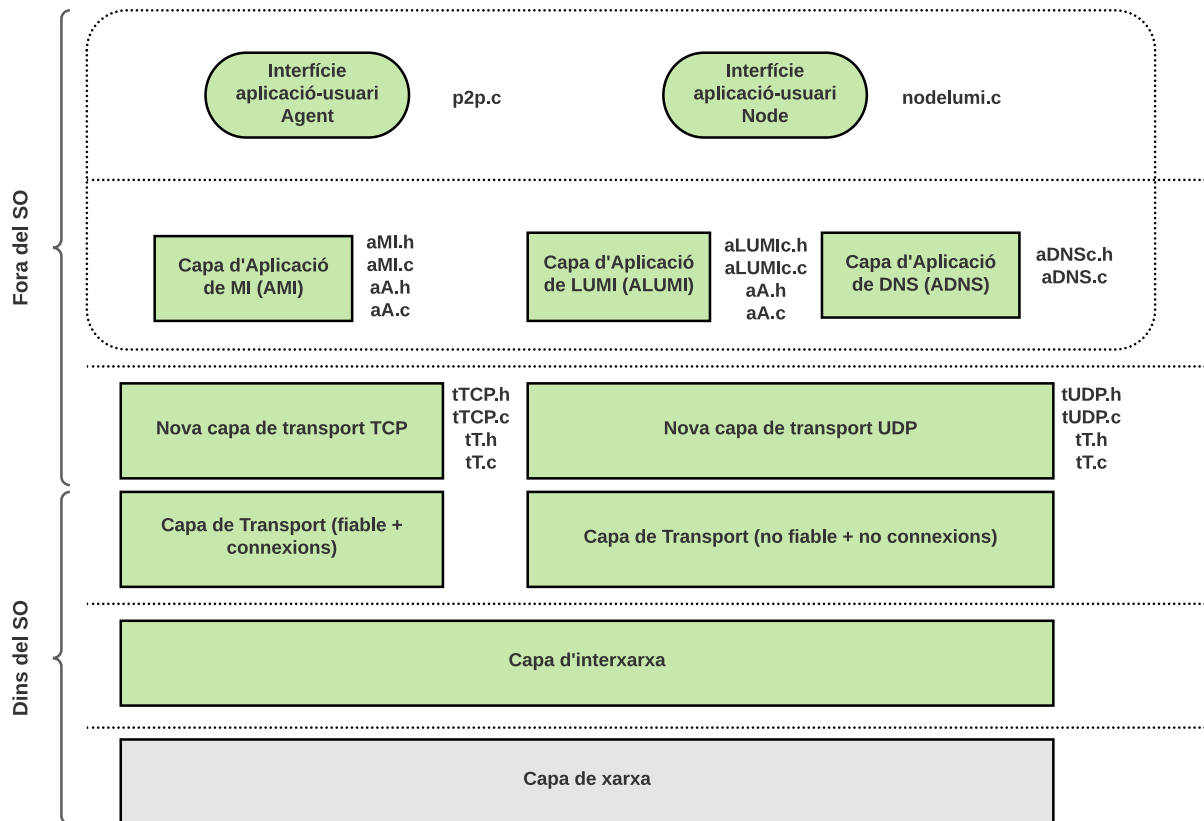
Maria diu: Que vagi bé!
Adéu!
/exit
-----
Prem ENTER per crear una connexió o ESPERA a una petició de connexió...
Entra EXIT per tancar el programa

```

## 3 Disseny de l'aplicació

### 3.1 L'arquitectura en capes

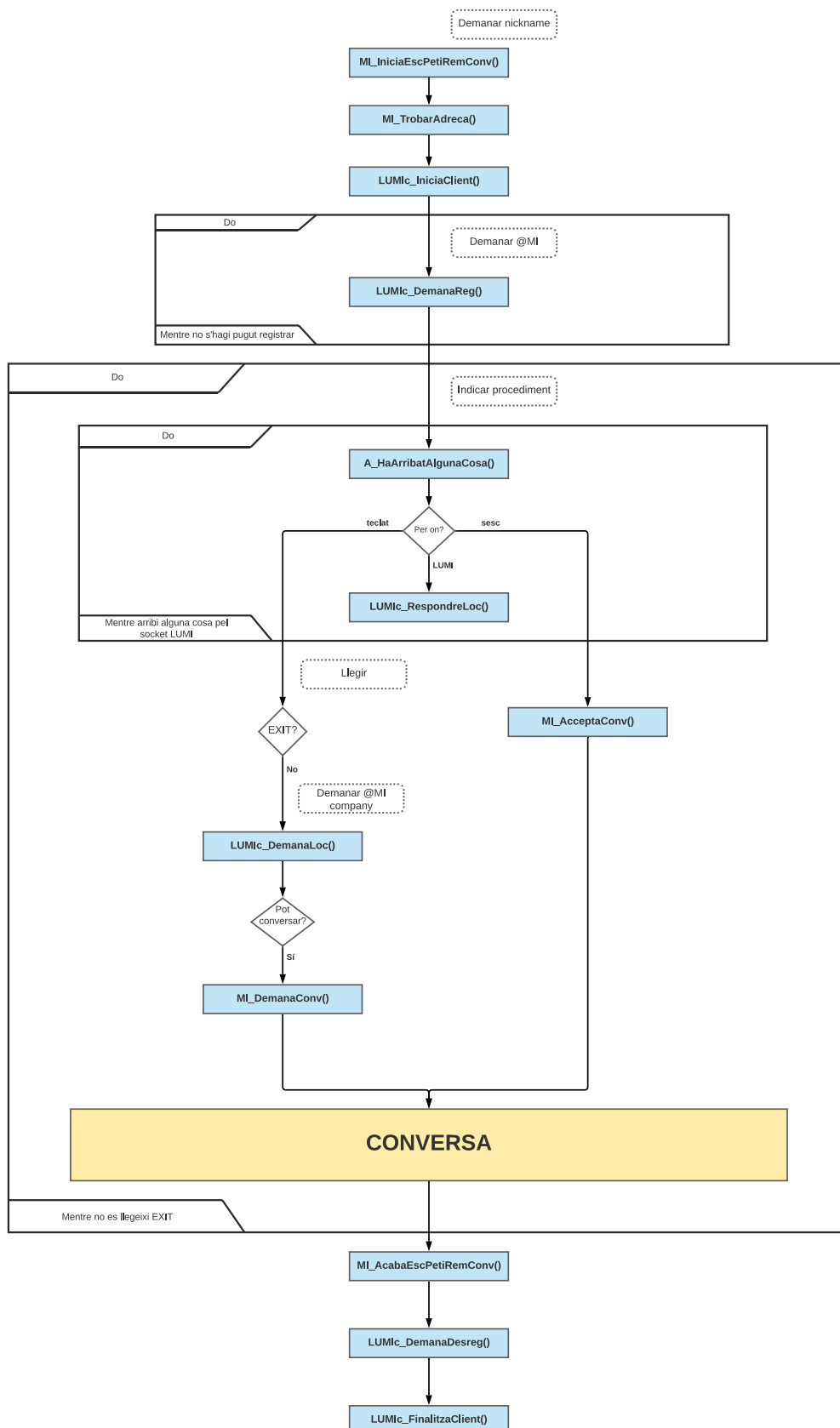


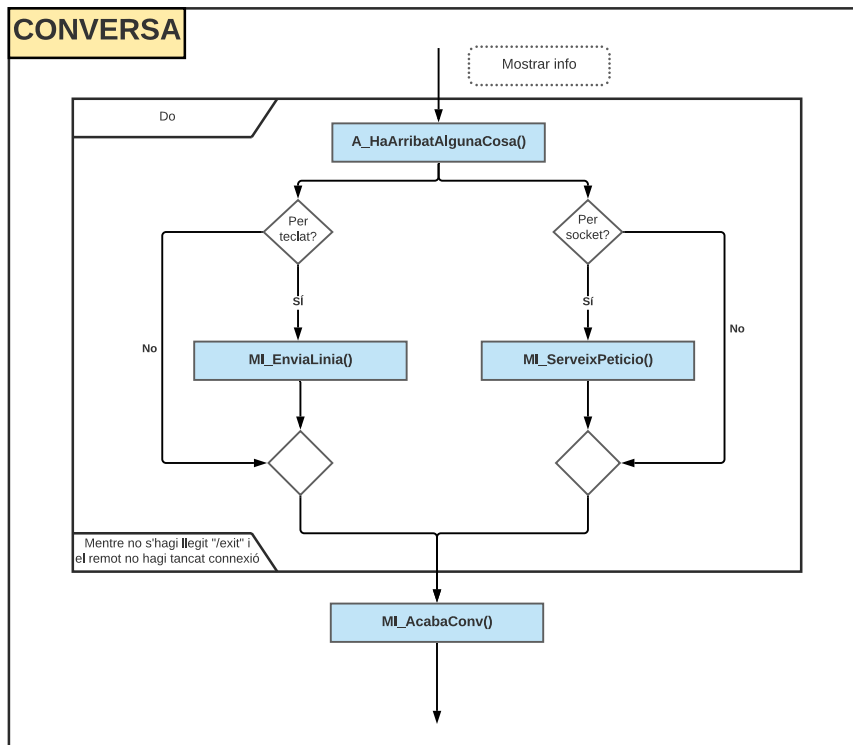


Nota: els fitxer aA i tT contenen funcions que comparteixen més d'una capa. Per aquesta raó hem decidit indicar quines capes els utilitzen.

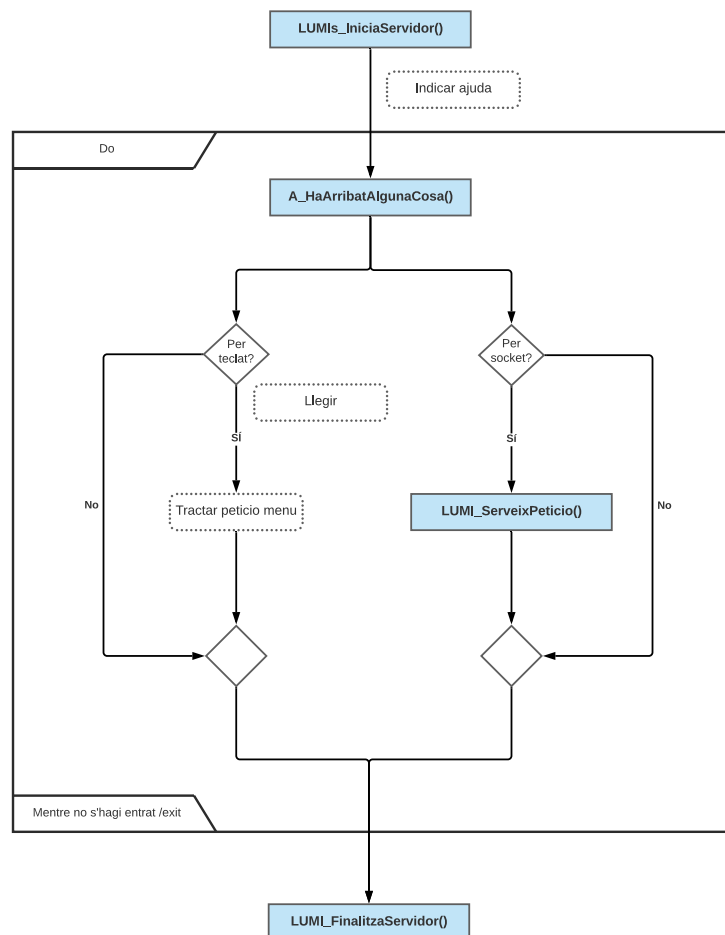
## 3.2 Diagrama funcional

### 3.2.1 Agent





### 3.2.2 Node



## 4 La interfície aplicació-usuari

Les funcions utilitzades per construir la interfície aplicació-usuari han estat les següents:

### Funcions d'interacció entre l'usuari i l'aplicació

- `int printf(const char *format, ...)` → imprimeix un missatge per pantalla utilitzant una cadena amb un format determinat. Cal incloure la llibreria `stdio.h`.
- `int scanf(const char *format, ...)` → permet llegir diversos tipus de dades d'una sola vegada, com ara enters, nombres decimals o cadenes de caràcters. Cal incloure la llibreria `stdio.h`.
- `ssize_t read(int fildes, void *buf, size_t nbyte)` → llegeix *nbytes bytes* de l'arxiu associat al *file descriptor* *fildes* i guarda el contingut al *buffer* apuntat per *buf*. Cal incloure la llibreria `unistd.h`.
- `FILE *popen(const char *command, const char *mode)` → Executa la comanda especificada a *command*. Crea una *pipe* entre el programa que crida i la comanda executada i retorna un punter a un *stream* que es pot utilitzar per llegir o escriure a la *pipe*. Cal incloure la llibreria `stdio.h`.
- `int pclose(FILE *stream)` → Tanca el *stream* que ha obert el `popen()`, espera que finalitzi el procés associat i retorna l'estat d'acabament. Cal incloure la llibreria `stdio.h`.
- `int fscanf(FILE *stream, const char *format, ...)` → Llegeix dades de *stream* i les guarda acord amb el paràmetre *format* a les variables addicionals. Cal incloure la llibreria `stdio.h`.
- `int fflush ( FILE * stream)` → Buida el *buffer* de sortida de *stream*. Cal incloure la llibreria `stdio.h`.

## 5 Les capes d'aplicació i transport (Serveis)

### 5.1 Serveis de la capa d'aplicació de MI

- Realitzar una conversa escrita en temps real entre dos usuaris.
- Escoltar peticions de conversa i acceptar les entrants.
- Sol·licitar una conversa.
- Intercanviar *nicknames* entre les dues estacions.
- Enviar i rebre missatges línia a línia.
- Tancar els *sockets*.

### 5.2 Serveis de la capa d'aplicació de LUMI

- Localitzar usuaris de MI, és a dir, traduir adreces MI a adreces de *socket* MI.
- Proporcionar la fiabilitat que li falta a la capa UDP.
- Oferir dues interfícies diferenciades, la de client i la de servidor.
- Permetre registres, desregistres i localitzacions.

### 5.3 Serveis de la capa d'aplicació de DNS

- Resoldre noms DNS a @IP.

### 5.4 Serveis de la capa de transport TCP

- Comunicar processos d'aplicació de diferents estacions de manera senzilla.
- Portar parquets entre processos d'aplicació.
- Crear *sockets*.
- Demanar i acceptar connexions.
- Trobar les adreces dels *sockets* local i remot.
- Proporcionar una connexió fiable i ordenada.

## 5.5 Serveis de la capa de transport UDP

- Comunicar processos d'aplicació de diferents estacions de manera senzilla.
- Portar paquets entre processos d'aplicació.
- Crear *sockets*.
- Mantenir el temps de transmissió el més baix possible.
- Permetre enviar i rebre paquets sense necessitat d'establir connexió.
- Proporciona una connexió no fiable, sense gestió de duplicats o paquets perduts.

## 6 Les capes d'aplicació i transport (Interfícies)

### 6.1 Interfície de la capa d'aplicació de MI

- `int MI_IniciaEscPetiRemConv(int portTCPloc) →` Inicia l'escolta de peticions remotes de conversa a través d'un nou socket TCP, que tindrà una @IP local qualsevol i el #port TCP "portTCPloc". Retorna -1 si hi ha error; l'identificador del socket d'escolta de MI creat si tot va bé.
- `int MI_DemanaConv(const char *IPrem, int portTCPrem, char *IPloc, int *portTCPloc, const char *NicLoc, char *NicRem) →` Crea un *socket* TCP client en un port i IP local qualsevol a través del qual es fa una petició de conversa a un procés remot. Omple IPloc i portTCPloc amb l'adreça IP i port TCP del *socket* del procés local. S'intercanvien els *nicknames*, omplint NickRem. Retorna l'identificador del *socket* de conversa o -1 si hi ha error.
- `int MI_AceptaConv(int SckEscMI, char *IPrem, int *portTCPrem, char *IPloc, int *portTCPloc, const char *NicLoc, char *NicRem) →` Accepta una petició de connexió remota rebuda pel *socket* SckEscMI i crea un *socket* de conversa. Omple IPloc, portTCPloc, IPrem i portTCPrem. S'intercanvien els *nicknames*, omplint NickRem. Retorna l'identificador del *socket* de conversa o -1 si hi ha error.
- `int MI_EnviaLinia(int SckConvMI, const char *Linia, int LongLinia) →` A través del *socket* de conversa SckConvMI, s'envia una línia amb la seva longitud. Retorna el nombre de caràcters de la línia enviada o -1 si hi ha error.
- `int MI_ServeixPeticio(int SckConvMI, char *SeqBytes, int *LongSeqBytes) →` A través del *socket* de conversa SckConvMI, s'até una petició de servei de l'usuari remot i omple SeqBytes i LongSeqBytes amb una informació que depèn del tipus de petició. Retorna 0 si s'acaba la conversa, 1 si rep una línia o -1 si hi ha error.
- `int MI_AcabaConv(int SckConvMI) →` Tanca la connexió del *socket* de conversa SckConvMI. Retorna -1 si hi ha error o un valor positiu qualsevol si tot va bé.
- `int MI_AcabaEscPetiRemConv(int SckEscMI) →` Tanca la connexió del *socket* d'escolta SckEscMI. Retorna -1 si hi ha error o un valor positiu qualsevol si tot va bé.
- `int MI_TrobarAdreca(int Sck, char * IPloc, int * portTCPloc) →` Troba l'adreça del *socket* Sck omplint IPloc i portTCPloc amb la seva IP i port. Retorna -1 si hi ha error o un valor positiu qualsevol si tot va bé.
- `char * MI_ObteMissError(void) →` Retorna un missatge de text que descriu l'error produït en la darrera crida de *sockets*.

## 6.2 Interfície de la capa d'aplicació de LUMI

- `int LUMIc_IniciaClient(int * Sck, int * fitxerLog, const char * nomFitxer)`  
→ Inicia el socket LUMI amb identificador `Sck` a través d'un nou socket UDP, que tindrà una IP i un port qualsevols. A la vegada, crea un fitxer de log amb nom `nomFitxer` (cadena de caràcters acabada en `\0` i longitud  $\leq 36$ ) i guarda el nou identificador del fitxer a `fitxerLog`. Retorna -1 si hi ha un error; 0 si tot va correctament.
- `int LUMIc_FinalitzaClient(int Sck, int fitxerLog)` → Tanca el socket LUMI amb identificador `Sck` i també el fitxer amb identificador `fitxerLog`. Retorna -1 si hi ha un error; 0 si tot va correctament.
- `int LUMIc_DemanaReg(int Sck, const char * adrecaMI, int fitxerLog)` → Demana una petició de registre de l'usuari amb adreca MI `adrecaMI` (cadena de caràcters acabada en `\0` i longitud  $\leq 37$ ) a través del socket LUMI `Sck`. Escriu al fitxer de log amb identificador `fitxerLog` una línia informativa sobre l'enviament de la petició de registre i, en cas de rebre una resposta, també n'escriu una de nova en el mateix fitxer. Retorna -1 si l'enviament ha donat error, -2 si el format de `adrecaMI` no és el correcte, -3 si el domini de `adrecaMI` no existeix; 0 si el registrament s'ha fet correctament o 1 l'usuari de la `adrecaMI` no està donat d'alta al seu domini. Retorna -1 si no s'ha trobat; la posició dins de la taula de registres en cas que existeixi.
- `int LUMIc_DemanaDesreg(int Sck, const char * adrecaMI, int fitxerLog)`  
→ Demana una petició de desregistre de l'usuari amb adreca MI `adrecaMI` (cadena de caràcters acaba en `\0` i longitud  $\leq 37$ , incloent `\0`) a través del socket LUMI amb identificador `Sck`. Escriu al fitxer de log amb identificador `fitxerLog` una línia informativa sobre l'enviament de la petició de registre i, en cas de rebre una resposta, també n'escriu una de nova en el mateix fitxer. Retorna -1 si l'enviament ha donat error; 0 si el desregistre s'ha fet correctament o 1 si l'usuari amb `adrecaMI` no existeix al seu domini.
- `int LUMIc_DemanaLoc(int Sck, const char * adrecaMI1, const char * adrecaMI2, char * IPremot, int * portremot, int fitxerLog)` → Demana una petició de localització de l'usuari amb adreca MI `adrecaMI2` (cadena de caràcters acabada en `\0` i longitud  $\leq 37$  incloent `\0`) per part de l'usuari amb adreca MI `adrecaMI1` (cadena de caràcters acabada en `\0` i longitud  $\leq 37$ , incloent `\0`) a través del socket LUMI amb identificador `Sck`. En cas que la localització es faci correctament, es a dir, es trobi l'usuari i aquest pugui establir una conversa, omple `IPremot` (string de C de mida màxima 16) i `portremot` amb la IP i el port de l'usuari amb `adrecaMI2`. Escriu al fitxer de log amb identificador `fitxerLog` una línia informativa sobre l'enviament de la petició de registre i, en cas de rebre una resposta, també n'escriu una de nova en el mateix fitxer. Retorna -1 si l'enviament de la petició ha donat error; 0 si s'ha realitzat la localització correctament i l'usuari destí està disponible per una conversa, 1 si l'usuari destí està ocupat en una altra conversa, 2 si l'usuari destí o el domini d'aquest no existeixen o 3 si l'usuari destí en aquests moments està *offline*.
- `int LUMIc_RespondreLoc(int Sck, int codi, const char * IPMI, const int * portMI, int fitxerLog)` → Respon una petició de localització a través del



socket LUMI amb identificador `Sck`. El camp `codi` indica si l'usuari local pot acceptar la petició de localització, per tant és 0, o ja està ocupat en una conversa, i llavors `codi` és 1. Els camps `IPMI` (string de C de mida màxima 16) i `portMI` són la IP i el port d'escolta de l'usuari local. Escriu al fitxer de log amb identificador `fitxerLog` una línia informativa sobre la petició de localització rebuda i a continuació, una nova línia amb la informació de la resposta a la petició. Retorna -1 si no s'ha pogut respondre a la localització; 0 si tot ha anat correctament.

- `char* LUMIc_ObteMissError(void)` → Retorna un missatge de text que descriu l'error produït en la darrera crida de sockets UDP (de la part client de LUMI).
- `int LUMIs_IniciaServidor(int * Sck, Registre * reg, int * fitxerLog, char * domini)` → Inicia el socket LUMI a través d'un nou socket UDP, amb IP qualsevol i port 3344 i guarda l'identificador del socket a `Sck`. També crea un fitxer de log amb nom "nodelumi-domini.log" i guarda el nou identificador del fitxer a `fitxerLog`. Alhora, llegeix el fitxer de configuració i carrega a `domini` el domini del servidor i a la taula de registres `reg` tots els usuaris que apareixen al fitxer, inicialitzats a *offline*. Retorna -1 si hi ha hagut un error; 0 si tot ha anat bé.
- `int LUMIs_FinalitzaServidor(const int Sck, const int fitxerLog, const Registre * reg)` → Sobreescriu els usuaris de la taula de registres `reg` al fitxer de configuració i tanca el socket LUMI amb identificador `Sck` i el fitxer de log amb identificador `fitxerLog`. Retorna -1 si hi ha un error; 0 si tot va bé.
- `int LUMIs_ServeixPeticio(const int Sck, Registre * reg, const int fitxerLog)` → Rep un paquet a partir del socket LUMI amb identificador `Sck` i el tracta segons el tipus de paquet: petició de localització, petició de registre, petició de desregistre i resposta de localització. A més, escriu al fitxer de log amb identificador `fitxerLog` una línia informativa sobre el paquet rebut i, en cas d'enviament, també n'escriu una de nova en el mateix fitxer. Segons el tipus de paquet rebut, modifica la taula de registres `reg` per tal de satisfer la sol·licitud rebuda. Retorna -1 si hi ha hagut un error; 0 si tot ha anat bé.
- `char* LUMIs_ObteMissError(void)` → Retorna un missatge de text que descriu l'error produït en la darrera crida de sockets UDP (de la part servidora de LUMI).

### 6.3 Interfície de la capa d'aplicació de DNS (part client)

- `int DNSc_ResolDNSaIP(const char *NomDNS, char *IP)` → Donat el nom DNS `NomDNS` s'obté la corresponent @IP i s'escriu a `IP`.
- `const char* DNSc_ObteMissError(void)` → Retorna un missatge de text que descriu l'error produït en la darrera crida de *sockets*.

### 6.4 Interfície de la capa de transport TCP

- `int TCP_CreaSockClient(const char *IPloc, int portTCPloc)` → Crea un *socket* TCP client amb IP `IPloc` i port `portTCPloc`. Retorna -1 si hi ha error; l'identificador del *socket* creat si tot va bé.

- `int TCP_CreaSockServidor(const char *IPloc, int portTCPloc) →` Crea un *socket* TCP servidor amb IP `IPloc` i port `portTCPloc`. Retorna -1 si hi ha error; l'identificador del *socket* creat si tot va bé.
- `int TCP_DemanaConnexio(int Sck, const char *IPrem, int portTCPrem) →` El *socket* client `Sck` es connecta al *socket* servidor de IP `IPrem` i port `portTCPrem`. Retorna -1 si hi ha error; un valor positiu si tot va bé.
- `int TCP_AceptaConnexio(int Sck, char *IPrem, int *portTCPrem) →` El *socket* servidor `Sck` accepta una connexió amb un *socket* client remot i es crea un nou *socket* de conversa. Omple `IPrem` amb la IP i `portTCPrem` amb el port del *socket* remot. Retorna -1 si hi ha error; l'identificador del *socket* de conversa si tot va bé.
- `int TCP_Envia(int Sck, const char *SeqBytes, int LongSeqBytes) →` Envia a través del *socket* `Sck` la seqüència de *bytes* escrita a `SeqBytes` (de longitud `LongSeqBytes`) cap al *socket* remot. Retorna -1 si hi ha error; el nombre de *bytes* enviats si tot va bé.
- `int TCP_Rep(int Sck, char *SeqBytes, int LongSeqBytes) →` Rep a través del *socket* `Sck` una seqüència de *bytes* que prové del *socket* remot i l'escriu a `SeqBytes` (de longitud `LongSeqBytes`). Retorna -1 si hi ha error, 0 si la connexió està tancada o el nombre de bytes rebuts si tot va bé.
- `int TCP_TancaSock(int Sck) →` S'allibera el *socket* `Sck` i, si estava connectat, es tanca la connexió. Retorna -1 si hi ha error; un valor positiu si tot va bé.
- `int TCP_TrobaAdrSockLoc(int Sck, char *IPloc, int *portTCPloc) →` Troba l'adreça del *socket* `Sck` omplint `IPloc` i `portTCPloc` amb la seva IP i port. Retorna -1 si hi ha error o un valor positiu qualsevol si tot va bé.
- `int TCP_TrobaAdrSockRem(int Sck, char *IPrem, int *portTCPrem) →` Donat el *socket* TCP connectat i identificat per `Sck`, troba l'adreça del *socket* remot amb qui està connectat, omplint `IPrem` i `portTCPrem` amb la seva IP i port. Retorna -1 si hi ha error o un valor positiu qualsevol si tot va bé.
- `char* TCP_ObteMissError(void) →` Retorna un missatge de text que descriu l'error produït en la darrera crida de *sockets*.

## 6.5 Interfície de la capa de transport UDP

- `int UDP_CreaSock(const char *IPloc, int portUDPloc) →` Crea un *socket* UDP amb IP `IPloc` i port `portUDPloc`. Retorna -1 si hi ha error; l'identificador del *socket* creat si tot va bé.
- `int UDP_EnviaA(int Sck, const char *IPrem, int portUDPrem, const char *SeqBytes, int LongSeqBytes) →` Envia a través del *socket* `Sck` la seqüència de *bytes* `SeqBytes` (de longitud `LongSeqBytes`) cap al *socket* remot que té IP `IPrem` i port `portUDPrem`. Retorna -1 si hi ha error; el nombre de *bytes* enviats si tot va bé.

- `int UDP_RepDe(int Sck, char *IPrem, int *portUDPrem, char *SeqBytes, int LongSeqBytes)` → Rep a través del *socket* `Sck` una seqüència de *bytes* que prové d'un *socket* remot i l'escriu a `SeqBytes` (de longitud `LongSeqBytes`). Omple `IPrem` i `portUDPrem` amb la IP i el port del *socket* remot. Retorna -1 si hi ha error; el nombre de *bytes* rebuts si tot va bé.
- `int UDP_TancaSock(int Sck)` → S'allibera el *socket* `Sck`. Retorna -1 si hi ha error; un valor positiu qualsevol si tot va bé.
- `int UDP_TrobaAdrSockLoc(int Sck, char *IPloc, int *portUDPloc)` → Troba l'adreça del *socket* `Sck`, omplint `IPloc` i `portUDPloc` amb la seva IP i port. Retorna -1 si hi ha error; un valor positiu qualsevol si tot va bé.
- `int UDP_DemanaConnexio(int Sck, const char *IPrem, int portUDPrem)` → El *socket* `Sck` es connecta al *socket* UDP remot amb IP `IPrem` i port `portUDPrem`. En el context de UDP això significa guardar l'adreça remota per permetre fer `Envia()` i `Rep()` en comptes de `EnviaA()` i `RepDe()`. Retorna -1 si hi ha error; un valor positiu qualsevol si tot va bé.
- `int UDP_Envia(int Sck, const char *SeqBytes, int LongSeqBytes)` → Envia a través del *socket* UDP `Sck` prèviament connectat la seqüència de *bytes* `SeqBytes` (de longitud `LongSeqBytes`) cap al *socket* remot amb qui està connectat. Retorna -1 si hi ha error; el nombre de bytes enviats si tot va bé.
- `int UDP_Rep(int Sck, char *SeqBytes, int LongSeqBytes)` → Rep a través del *socket* UDP `Sck` prèviament connectat una seqüència de *bytes* que prové del *socket* remot amb qui està connectat i l'escriu a `SeqBytes` (de longitud `LongSeqBytes`). Retorna -1 si hi ha error; el nombre de *bytes* rebuts si tot va bé.
- `int UDP_TrobaAdrSockRem(int Sck, char *IPrem, int *portUDPrem)` → Donat el *socket* UDP `Sck` connectat, troba l'adreça del *socket* remot amb qui està connectat. Omple `IPrem` i `portUDPrem` amb la seva IP i port UDP. Retorna -1 si hi ha error; un valor positiu qualsevol si tot va bé.
- `char* UDP_ObteMissError(void)` Retorna el missatge produït a la darrera crida de *sockets* UDP.

## 6.6 Part compartida de TCP i UDP

- `int T_HaArribatAlgunaCosa(const int *LlistaSck, int LongLlistaSck)` → Examina simultàniament i sense límit de temps els *sockets* amb identificadors a la llista `LlistaSck` (de longitud `LongLlistaSck`) per saber si ha arribat alguna cosa per a ser llegida. Retorna l'identificador del *socket* pel que ha arribat alguna cosa o -1 si hi ha error.
- `int T_HaArribatAlgunaCosaEnTemps(const int *LlistaSck, int LongLlistaSck, int Temps)` → Examina simultàniament i amb límit de temps `Temps` (ms) els *sockets* amb identificadors a la llista `LlistaSck` (de longitud `LongLlistaSck`) per saber si ha arribat alguna cosa per a ser llegida. Si `Temps` és -1 s'espera indefinidament. Retorna l'identificador del *socket* pel que ha arribat alguna cosa, -1 si hi ha error o -2 si expira el *timeout*.

- `char* T_ObteMissError(void)` → Retorna el missatge produït a la darrera crida de *sockets*.

## 6.7 Part compartida de MI i LUMI

- `int A_HaArribatAlgunaCosa(const int *LlistaSck, int LongLlistaSck)` → Examina simultàniament i sense límit de temps els *sockets* amb identificadors a la llista *LlistaSck* (de longitud *LongLlistaSck*) per saber si ha arribat alguna cosa per a ser llegida. Retorna l'identificador del *socket* pel que ha arribat alguna cosa o -1 si hi ha error.
- `char* A_MostraError(void)` → Retorna el missatge produït a la darrera crida de *sockets*.

## 7 Les capes d'aplicació i transport (Protocols)

### 7.1 Protocol de les capes d'aplicació de MI (PMI) i de LUMI (PLUMI)

El protocol **PMI** es pot definir de la següent manera:

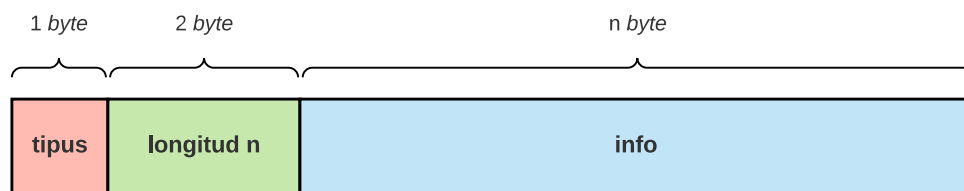
- **Nom i significat**

- PMI\_LIN o missatge de LÍNia, que transporta una línia de la conversa.
- PMI\_NIC o missatge de NICKname, que transporta el *nickname* d'un usuari.

Per saber si un procés remot vol iniciar o acabar una connexió no hi ha un missatge definit, sinó que es fa servir la mateixa connexió i desconnexió TCP per indicar-ho.

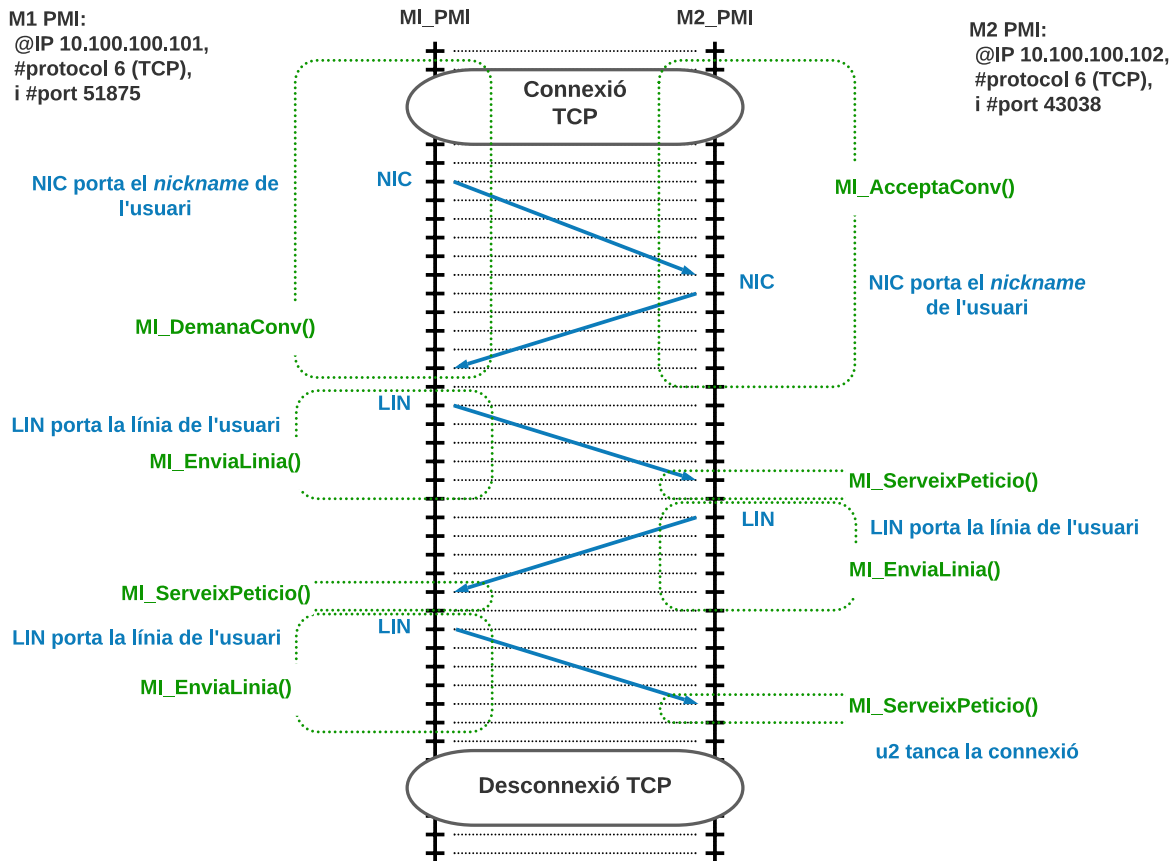
- **Format**

El missatge de línia PMI\_LIN i el de nickname PMI\_NIC tenen tres camps i la longitud màxima del paquet és 102 *bytes*:



- Camp “tipus”: conté el caràcter ‘L’ o ‘N’ (de tipus char en C) per indicar el missatge PMI\_LIN o PMI\_NIC respectivament.
- Camp “longitud n”: conté la longitud en *bytes* del camp “informació”, escrita en forma de caràcters.
- Camp “informació”: conté la informació del missatge. En el cas d'un PMI\_LIN, conté una línia escrita, i en el cas d'un PMI\_NIC conté un *nickname*. Ambdós casos no contenen final de línia ( $\backslash n$ ) ni fi de *string* ( $\backslash 0$ ). La longitud del camp és  $0 \leq n \leq 99$ .

- **Seqüència Temporal**



El protocol **LUMI** és pot definir de la següent manera:

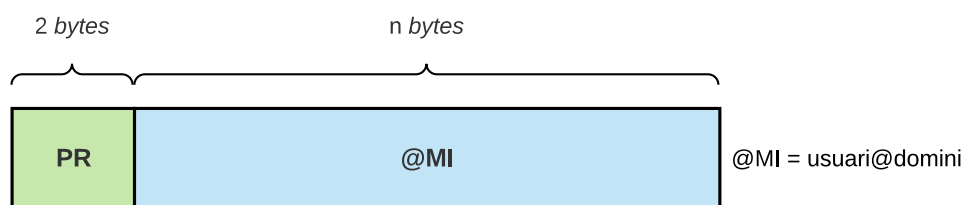
- **Nom i significat**

- PR: Petició de registre de l'usuari @MI (passa a estar *online*).
- RR: Resposta a la petició de registre d'un usuari.
- PD: Petició de desregistre de l'usuari @MI (passa a estar *offline*).
- RD: Resposta a la petició de desregistre d'un usuari.
- PL: Petició de Localització de l'usuari @MI2 (demanada per l'usuari @MI1)
- RL: Resposta a la petició de Localització de l'usuari @MI2 (demanada per l'usuari @MI1)

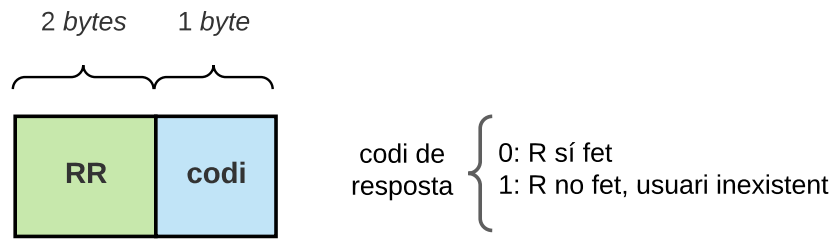
- **Format**

La longitud màxima del paquet és 100 i el format depén del tipus de paquet. Tots comparteixen 2 bytes de tipus. A continuació es llisten:

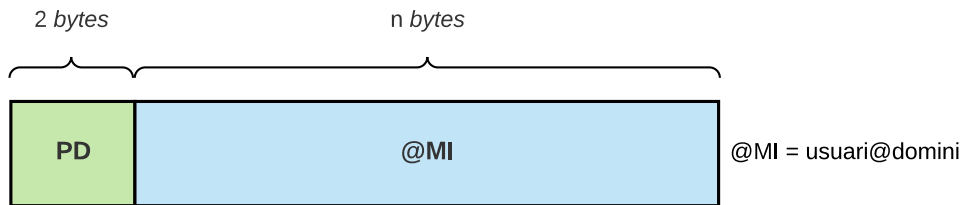
- PR



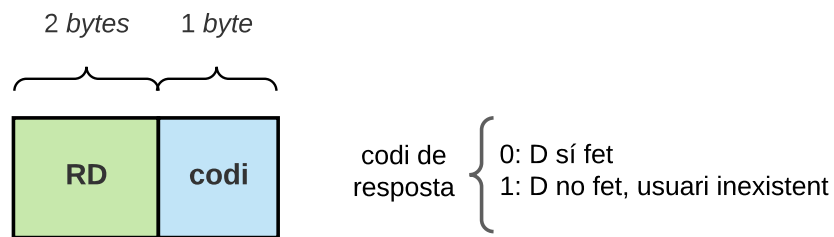
– RR



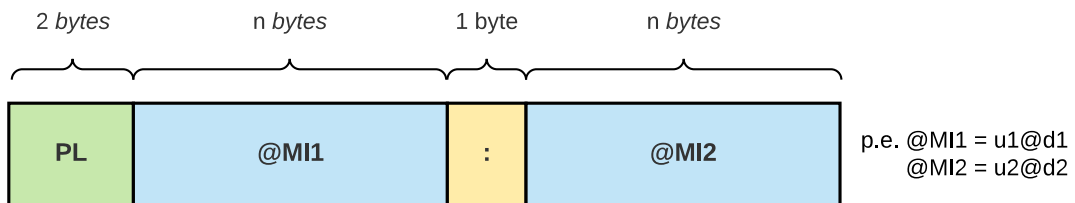
– PD



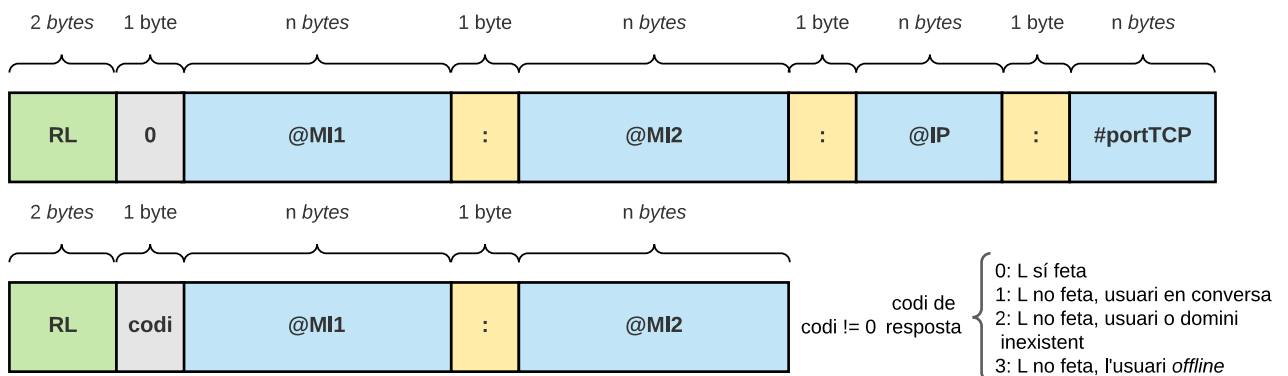
– RD



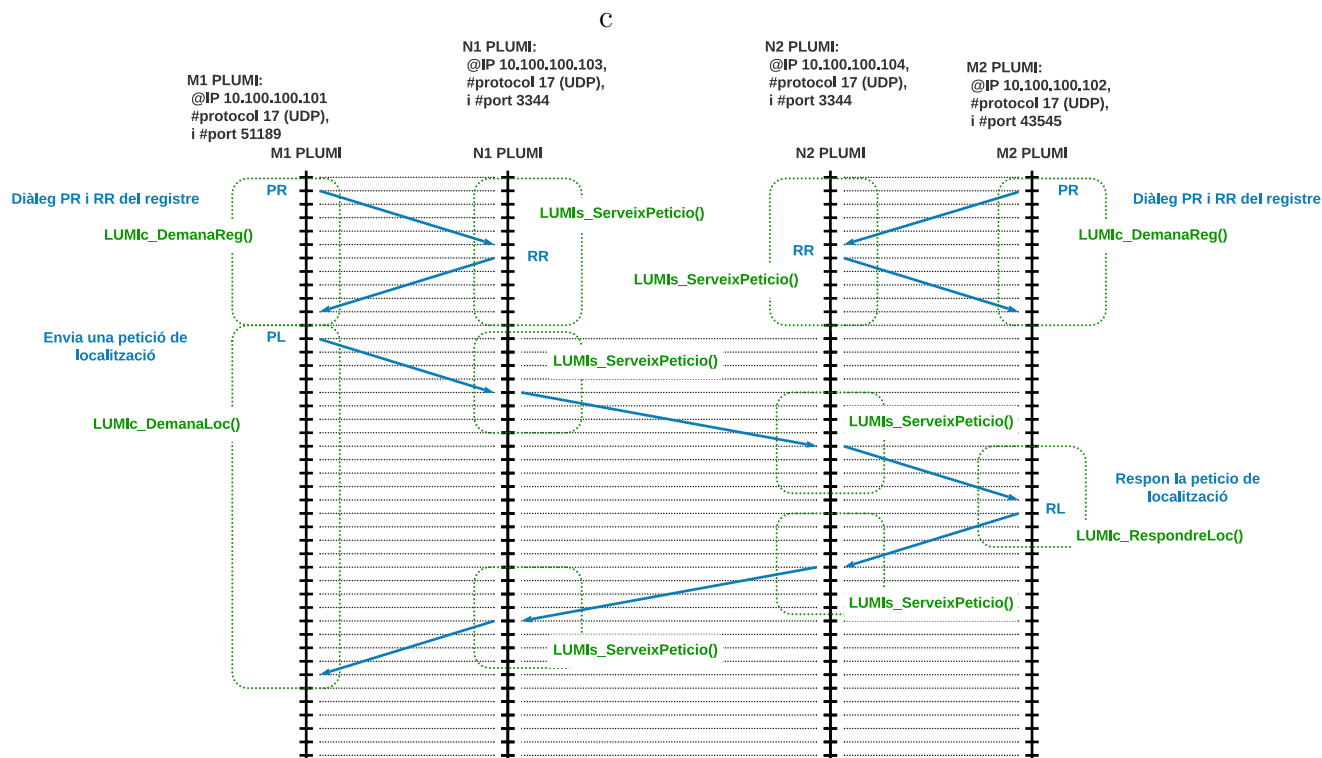
– PL



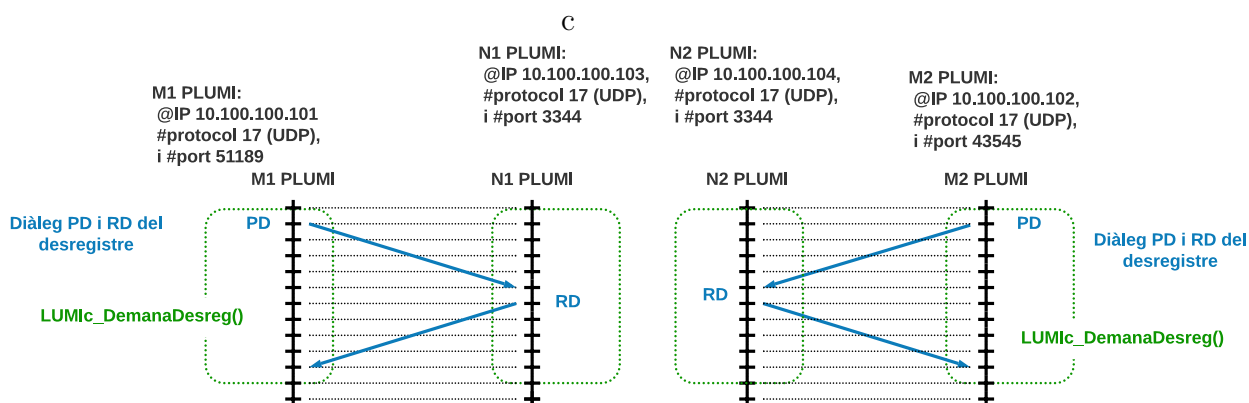
– RL



## • Seqüència temporal



[Aquí és on es realitza la conversa, tal com mostra el diagrama de seqüència de PMI.]



## 7.2 Protocol de les capes de transport TCP i UDP.

El protocol **TCP** és pot definir de la següent manera:

- **Nom i significat**

- Els missatges de petició d'inici de connexió i resposta (PIC, RP) són els paquets SYN, SYN+ACK i ACK.
- Els missatges de petició de fi de connexió i resposta (PFC, RP), són els paquets FIN, FIN+ACK i ACK.
- El missatge de confirmació positiva (BEN) és un paquet "ACK sense info", és a dir, un ACK que no porta informació i és acumulatiu.

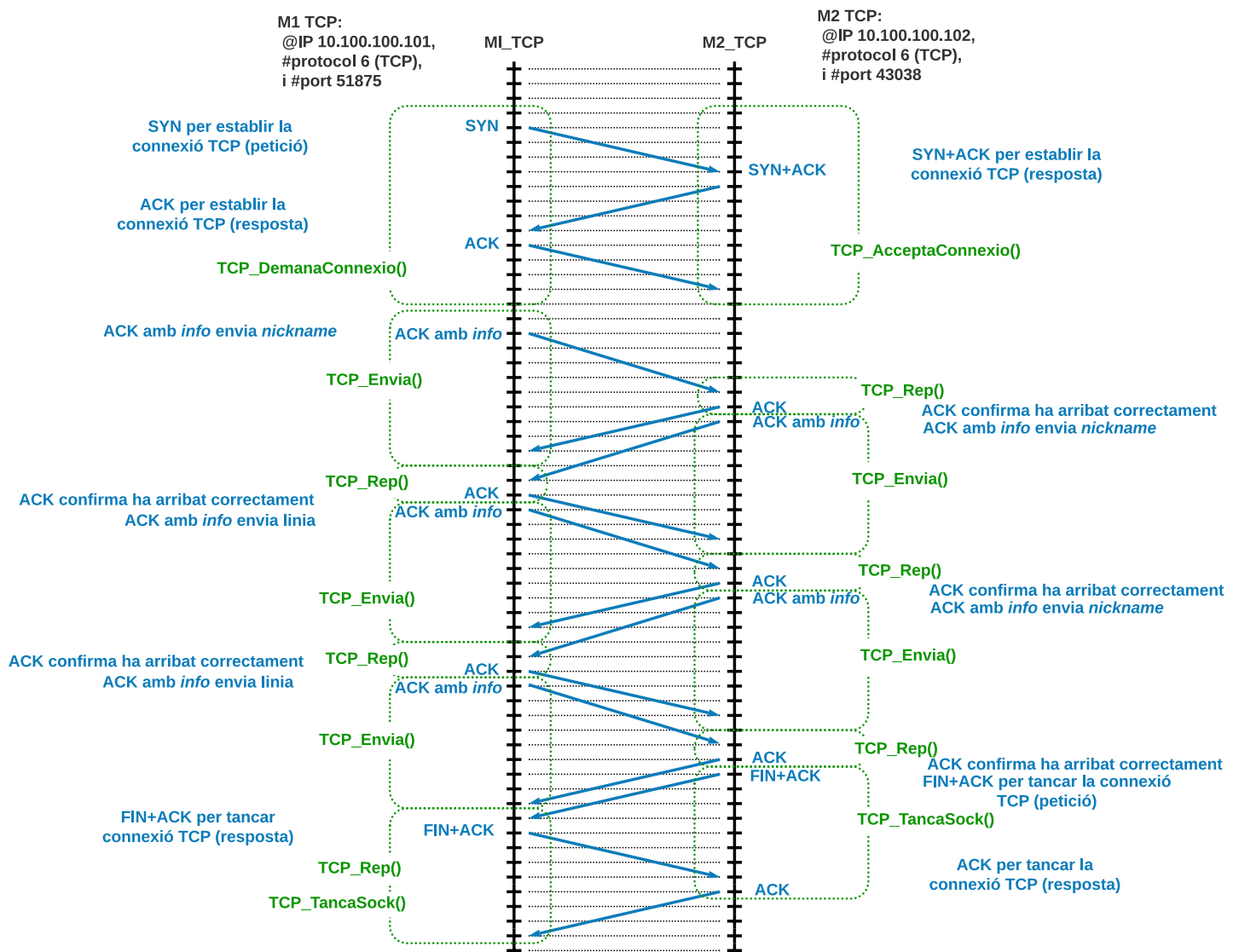


- El missatge d'informació (I) és un paquet “ACK amb info” (que alhora és també una confirmació positiva BEN en l'altre sentit); el flag PSH és “secundari” i a vegades acompanya els paquets “ACK amb info”.

- **Format**

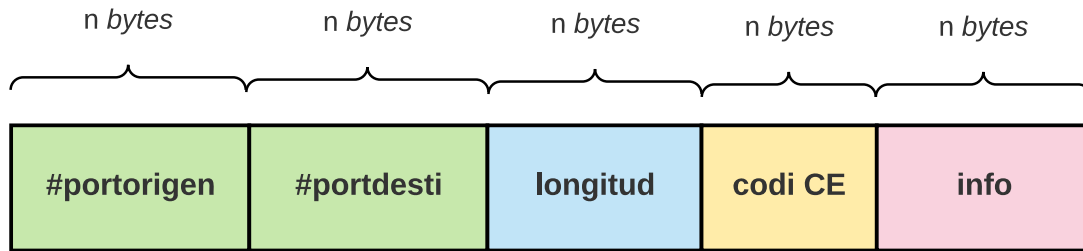
Un paquet TCP conté els següents camps: El port origen, el port destí, el número de seqüència, els *flags* i altres camps de control d'errors i informació. El camp de *flags* indica el tipus de missatge.

- **Seqüència Temporal**

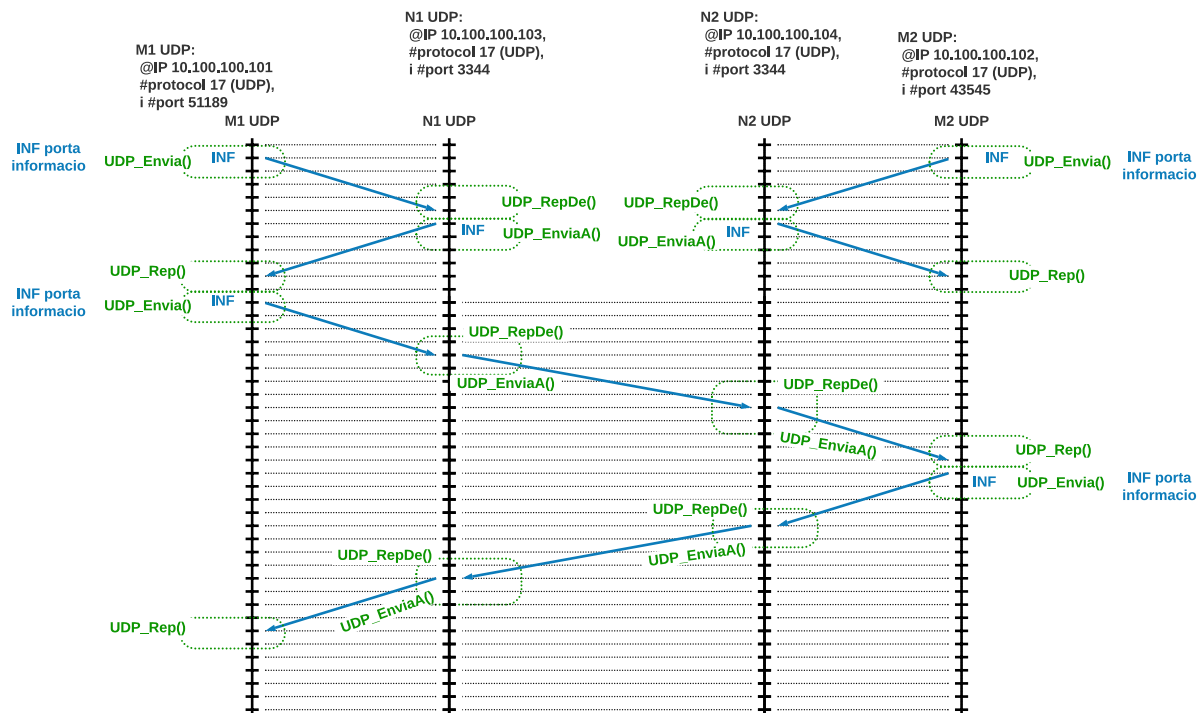


El protocol **UDP** és pot definir de la següent manera:

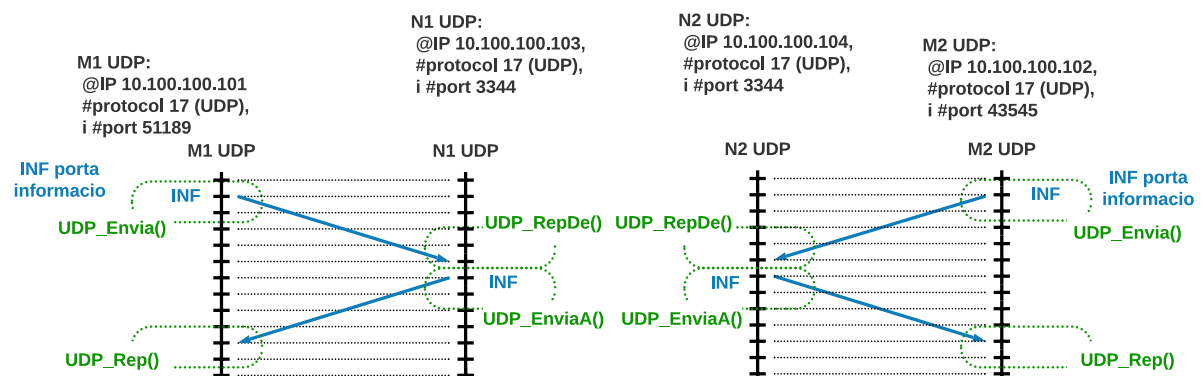
- **Nom i significat** El protocol UDP només treballa amb un únic missatge:
  - INF: missatge que porta informació.
- **Format** El format del paquet INF és el següent:



### • Seqüència temporal

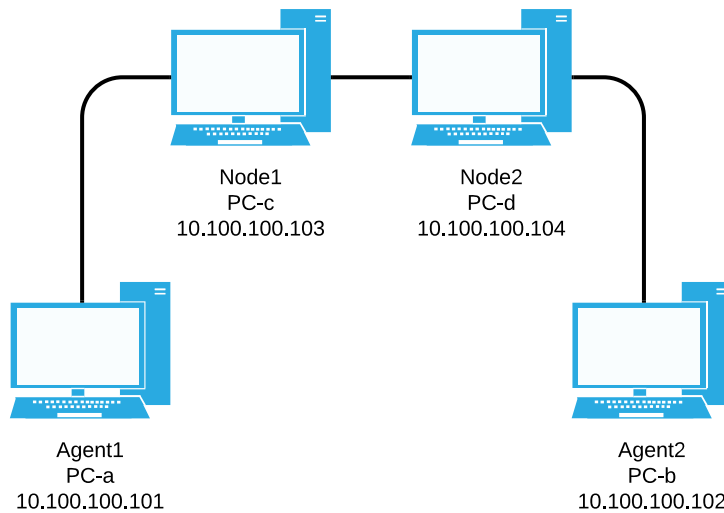


[Aquí és on es realitza la conversa, tal com mostra el diagrama de seqüència de TCP.]



## 7.3 Estudi amb “ss”

Per realitzar l'estudi, hem connectat 4 màquines de la següent manera:



A l'*Agent1* hi ha l'usuari *u1* i a l'*Agent2* hi ha l'usuari *u2*, prèviament donats d'alta al *Node1* i *Node2*, respectivament.

A continuació es mostra l'execució realitzada als dos agents:

```

xarxes@UbuntuX1:~/Desktop/P2$ ./aplicacio
-----
Client d'aplicació MI
-----
LOGIN
Introdueix el teu nick (màxim 20 caràcters):
> u1
Entra @MI (mida MAX 36 caràcters) nomMI@domini:
> u1@PC-c
S'ha registrat correctament al node
-----
Prem ENTER per crear una connexió o ESPERA a una petició de connexió...
Entra EXIT per tancar el programa
-----
Entra @MI destí (mida MAX 36 caràcters) nomMI@domini:
> u2@PC-d
S'ha pogut localitzar al company de conversa
-----
Nickname Local -> u1
Nickname Remot -> u2
-----
ADREÇA LOCAL -> 10.100.100.101:51875
ADREÇA REMOTA -> 10.100.100.102:43038
-----
Hola u2!
u2 diu: Hola! Que tal?
Be, haig de marxar, adeu
/exit
-----
Prem ENTER per crear una connexió o ESPERA a una petició de connexió...
Entra EXIT per tancar el programa
EXIT
-----
T'has desregistrat del teu domini
Fi execució
xarxes@UbuntuX1:~/Desktop/P2$

```

*Agent1*

```

xarxes@UbuntuX1:~/Desktop/P2$ ./aplicacio
-----
Client d'aplicació MI
-----
LOGIN
Introdueix el teu nick (màxim 20 caràcters):
> u2
Entra @MI (mida MAX 36 caràcters) nomMI@domini:
> u2@PC-d
S'ha registrat correctament al node
-----
Prem ENTER per crear una connexió o ESPERA a una petició de connexió...
Entra EXIT per tancar el programa
-----
Nickname Local -> u2
Nickname Remot -> u1
-----
ADREÇA LOCAL -> 10.100.100.102:43038
ADREÇA REMOTA -> 10.100.100.101:51875
-----
u1 diu: Hola u2!
Hola! Que tal?
u1 diu: Be, haig de marxar, adeu
-----
Prem ENTER per crear una connexió o ESPERA a una petició de connexió...
Entra EXIT per tancar el programa
EXIT
-----
T'has desregistrat del teu domini
Fi execució
xarxes@UbuntuX1:~/Desktop/P2$

```

*Agent2*

Utilitzant la comanda `ss -natu` hem obtingut una llista dels *sockets* actius a les dues màquines:

```

xarxes@UbuntuX1:~$ ss -natu
Netid State Recv-Q Send-Q Local Address:Port Peer Address:Port
udp ESTAB 0 0 10.100.100.101:51189 10.100.100.103:3344
tcp LISTEN 0 3 *:59966 *:59966
tcp ESTAB 0 0 10.100.100.101:51875 10.100.100.102:43038
xarxes@UbuntuX1:~$

```

*Agent1*

```

xarxes@UbuntuX1:~$ ss -natu
Netid State Recv-Q Send-Q Local Address:Port Peer Address:Port
udp ESTAB 0 0 10.100.100.102:43545 10.100.100.104:3344
tcp LISTEN 0 3 *:43038 *:43038
tcp ESTAB 0 0 10.100.100.102:43038 10.100.100.101:51875
xarxes@UbuntuX1:~$

```

*Agent2*

Com es pot veure a les captures anteriors, a l'*Agent1* hi ha un *socket* UDP “connectat” al seu domini, és a dir, al *Node1*, pel port 3344, que és el que defineix el protocol per a connexions entre agents i nodes LUMI. L'*Agent2* també té un *socket* similar, tot i que

connectat al *Node2*. En els paquets capturats amb el *Wireshark*, es pot veure com també coincideixen les IP i ports origen i destí.

El segon *socket* és un de TCP en estat d'escolta, pertanyent a la part P2P de l'aplicació. Tots dos agents el tenen actiu, escoltant a qualsevol IP.

Per últim, hi ha el *socket* TCP de conversa, que està connectat entre els agents. Tant al primer com al segon agent, les IP i ports coincideixen. Això també passa amb les IP i ports que es mostren per pantalla quan s'inicia la conversa i amb les adreces dels paquets capturats.

També hem obtingut la llista de *sockets* dels dos nodes i la sortida per pantalla dels dos és idèntica. A continuació és mostra aquesta llista:

```
xarxes@UbuntuX1:~$ ss -natu
Netid  State      Recv-Q  Send-Q    Local Address:Port    Peer Address:Port
udp    UNCONN    0        0         *:3344          *:*
```

A la imatge anterior es pot veure el *socket* UDP que tenen els nodes per servir peticions LUMI. El seu estat és desconnectat perquè no es “connecta” a cap agent en concret. D’una manera similar al *socket sesc*, està escoltant peticions a qualsevol IP de la màquina, però en el port 3344 i per peticions LUMI. Als paquets capturats dels flux entre agents i nodes es veu com el port dels nodes coincideix.

## 8 Problemes i suggeriments

Durant el desenvolupament de la pràctica ens hem trobat amb alguns problemes. Alguns d'ells eren per no utilitzar correctament funcions de llibreries de C, com és `strtok()`, o pel tractament de *strings* en C. Tanmateix, no van ser complicats de detectar i solucionar.

Respecte a la capa de transport, ens vam trobar amb un error difícil de detectar i és que la funció `T_HaArribatAlgunaCosaEnTemps` ens mostrava un error de “Invalid Argument”. Sembla ser que és necessari inicialitzar l'estructura `timeval tv` a 0.

Un altre error que ens va costar detectar és que a la funció `obtenirIPportEsc` utilitzàvem la funció `fgets()`. No ens vam adonar que li passàvem com a paràmetre el `sizeof(IPescolta)`, però `IPescolta` és un punter a enter. Per tant, el punter es degrada a la primera posició i mai se'ns carregava la IP correctament. Vam decidir substituir el `fgets()` per un `fscanf()`.

A l'hora de clonar més màquines virtuals per fer proves, utilitzàvem l'opció de generar noves adreces MAC per tots els adaptadors de xarxa. Tot i això, el sistema operatiu conservava la MAC de la màquina original. Per solucionar-ho, vam esborrar el fitxer que feia referència a l'antic adaptador de xarxa per tal que, un cop es reiniciés la màquina, es tornés a generar. Així la màquina tindria l'adaptador de xarxa ben configurat i preparat per connectar-se.

En el nostre cas no tenim cap suggeriment per fer de la pràctica.

## 9 Treball en parella i dedicació

Per realitzar la pràctica hem treballat de manera conjunta i equitativa. Gràcies a una extensió del *clion*, ens podíem connectar mútuament a una sessió i programa a la vegada. D'aquesta manera, tots dos ens hem involucrat en el projecte, buscant informació i organitzant el codi. A l'hora de trobar solucions a errors, cadascú ho feia pel seu compte perquè porta temps i així posteriorment ho podíem discutir. En algun moment si que programàvem per separat, per exemple, la Laura va fer el registre i desregistre mentre que en David va fer dues de les millores proposades. En canvi, la petició de localització, la resposta de localització o l'organització de les interfícies les vam fer plegats. En definitiva, podem dir que ambdós hem treballat en la mateixa mesura i hem entès totes les parts del projecte.

Les hores dedicades han variat en funció del deures setmanals, que hem intentat sempre porta al dia. També s'ha de dir que després de fer la pràctica 1, ja estàvem més acostumats a treballar amb *sockets* i a lidiar amb els errors. Podem dir que de manera general cada setmana cadascú ha dedicat a la pràctica unes 7 hores. Per tant, si sumem les 8 hores dedicades al informe i hores extres per solucionar errors, cadascú ha dedicat aproximadament 50 hores a la pràctica.