UNIVERSITAT DE BARCELONA

Pràctica 3. Comunicacions a través de la pila TCP/IP

Óscar Jiménez, NIUB: 20100286
Laurentiu Nedelcu, NIUB: 20081585

ÍNDEX

OBJECTIUS

L'objectiu principal d'aquesta tercera pràctica és conèixer el funcionament del protocol TCP/IP i treballar sobre aquesta pila per muntar un sistema de transferència de dades entre un dispositiu i un servidor, analitzant com són els paquets necessaris.

Per tal d'assolir aquests objectius haurem de crear un canal usant ThingSpeak, el qual ens serviria com a recol·lector de dades, on enviarem la informació des del nostre dispositiu (mota). A més, haurem de crear un programa on la nostra mota generi una xarxa WiFi amb la qual ens podem connectar i transmetre informació. Per finalitzar, per tal d'analitzar els diferents protocols usarem un software de simulació (Packet Tracer de Cisco) que ens permetrà veure l'estructura dels diferents paquets que estem enviant per la xarxa.

PREGUNTES I RESPOSTES

```
#include <ESP8266WiFi.h>

// Wi-Fi Settings
const char* ssid = "XXX"; // your wireless network name (SSID)
const char* password = "XXXXXXXX"; // your Wi-Fi network password

WiFiClient client;

// ThingSpeak Settings
const int channelID = XXX;
String writeAPIKey = "XXXXXXXXXXXXXXXX"; // write API key for your ThingSpeak Channel
const char* server = "api.thingspeak.com";
const int postingInterval = 20 * 1000; // post data every 20 seconds
```

Imatge 1. Conté el codi amb les variables locals i les llibreries incloses.

```
void loop() {
  if (client.connect(server, 80)) {

    // Measure Signal Strength (RSSI) of Wi-Fi connection
    long rssi = WiFi.RSSI();

    // Construct API request body
    String body = "field1=";
           body += String(rssi);

    Serial.print("RSSI: ");
    Serial.println(rssi);

    client.println("POST /update HTTP/1.1");
    client.println("Host: api.thingspeak.com");
    client.println("User-Agent: ESP8266 (nothans)/1.0");
    client.println("Connection: close");
    client.println("X-THINGSPEAKAPIKEY: " + writeAPIKey);
    client.println("Content-Type: application/x-www-form-urlencoded");
    client.println("Content-Length: " + String(body.length()));
    client.println("");
    client.print(body);

  }
```

Imatge 2. Codi amb el loop que transmet la dada RSSI al servidor.

Expliqueu detalladament el funcionament del codi.

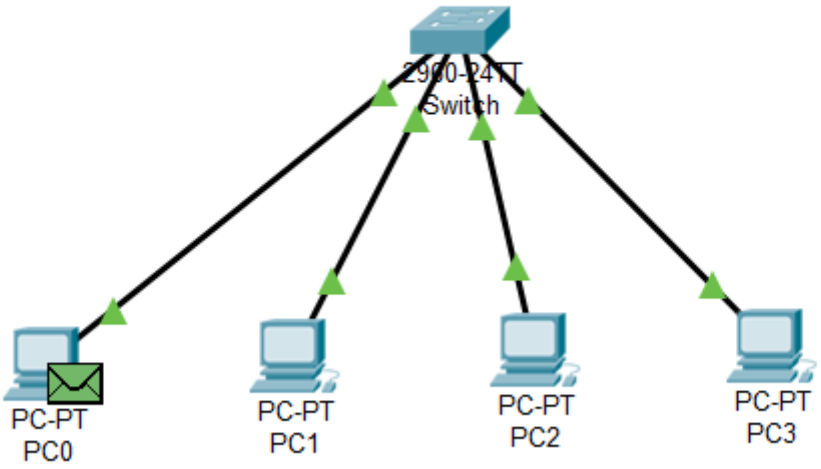
Primerament incloem la llibreria “ESP8266WiFi.h” la qual ens proporciona les funcions necessàries per tal de connectar el nostre mòdul a una xarxa WiFi per començar a enviar i rebre dades. Després posem unes variables globals els quals consisteixen en:

- Dues cadenes de text ssid i password, que com hem vist en la pràctica anterior contindran el nom de la WiFi a la qual ens volem connectar i la seva contrasenya.
- El WiFiClient que ens crea un client que es pot connectar a una adreça IP d’Internet i el port amb la funció client.connect(), la qual usarem més tard.
- Per tal de poder comunicar-nos amb ThingSpeaks necessitarem el id del canal (channelID), la clau API (writeAPIKey) la qual conté un codi únic que se li passa a l’API per a identificar l’aplicació o l’usuari que el crida, el servidor amb el qual ens connectarem (server) i una variable (postingInterval) que ens donarà un retard de 20 segons quan el cridéssim amb la funció delay(). Aquesta última ens serà útil per a que el servidor tingui el suficient temps per a processar les dades cada vegada que les enviéssim.

Seguidament entrem en el loop() on en primer lloc ens trobem amb aquesta funció esmentada anteriorment, client.connect(). Aquesta es connecta a una adreça IP i a un port especificat. Es pot especificar un port en el rang de 0-65536 al servidor NCSA. Per defecte, el número de port per a un servidor web és 80 que és el que estem usant. El nostre servidor serà l'aplicació ThingSpeak. La funció ens retornarà un valor sencer, que segons la pàgina d'arduino són: èxit (1), s'ha acabat el temps (-1), servidor no vàlid (-2), truncada (-3) i resposta invàlida (-4). Tot i això, també ens pot retornar 0 en cas de que no ens hàgim pogut connectar al servidor i **normalment** sol donar-nos aquest error quan no hem configurat correctament el nostre dispositiu.
Després ens trobem amb una funció ja familiaritzada, el WiFi.RSSI(). Aquesta ens retorna la intensitat de la senyal rebuda i serà la nostra dada a enviar. Aquesta dada la guardem en el camp 1 del nostre "body", aquest String serà el que enviarem al nostre servidor. Tal com s'explica en la pràctica, quan hem creat el nostre canal a ThingSpeak hem declarat que el camp 1 contindrà la RSSI.
El serial.print() tan sols ens mostra a nosaltres el valor de la RSSI al nostre monitor.
Usant el client.println(POST) li estem dient al nostre servidor que li enviarem dades per a processar amb el nostre dispositiu. Mitjançant el client.println(HOST) especifiquem que el nostre host serà el ThingSpeak. El client.println(user-agent) és usat pel servidor remot per provar i determinar quin és el navegador que usem per a oferir-nos la millor experiència visual. Usem el client.println(Connection: close) ja que el HTTP / 1.1 defineix la opció de connexió "close" per al remitent de la senyal que es tancarà la connexió quan s'hagi completat la sessió de resposta. Per a més informació visiteu el següent enllaç: https://tools.ietf.org/html/rfc2616#section-14.10. Seguidament, escrivim la clau API (la qual hem explicat anteriorment) que ens proporciona ThingSpeak. Abans, hem vist que el HTTP POST envia les dades al servidor. El tipus de cos de la sol·licitud es indicada per la capçalera Content-Type. Usant el client.println("Content-Type: application/x-www-form-urlencoded") els valors són codificats en tuples de clau-valor separats per un '&', la separació clau-valor es fa mitjançant '='. Els caràcters no-alfanumèrics en ambdues clau-valor són percent encoded. Per tant, no es adequat usarse amb dades binaries (s'hauria d'usar multipart/form-data). Per finalitzar posem la mida de les dades que enviarem en Content-Length i enviem el "body" amb client.print per a que el servidor no llegeixi també el salt de línia.

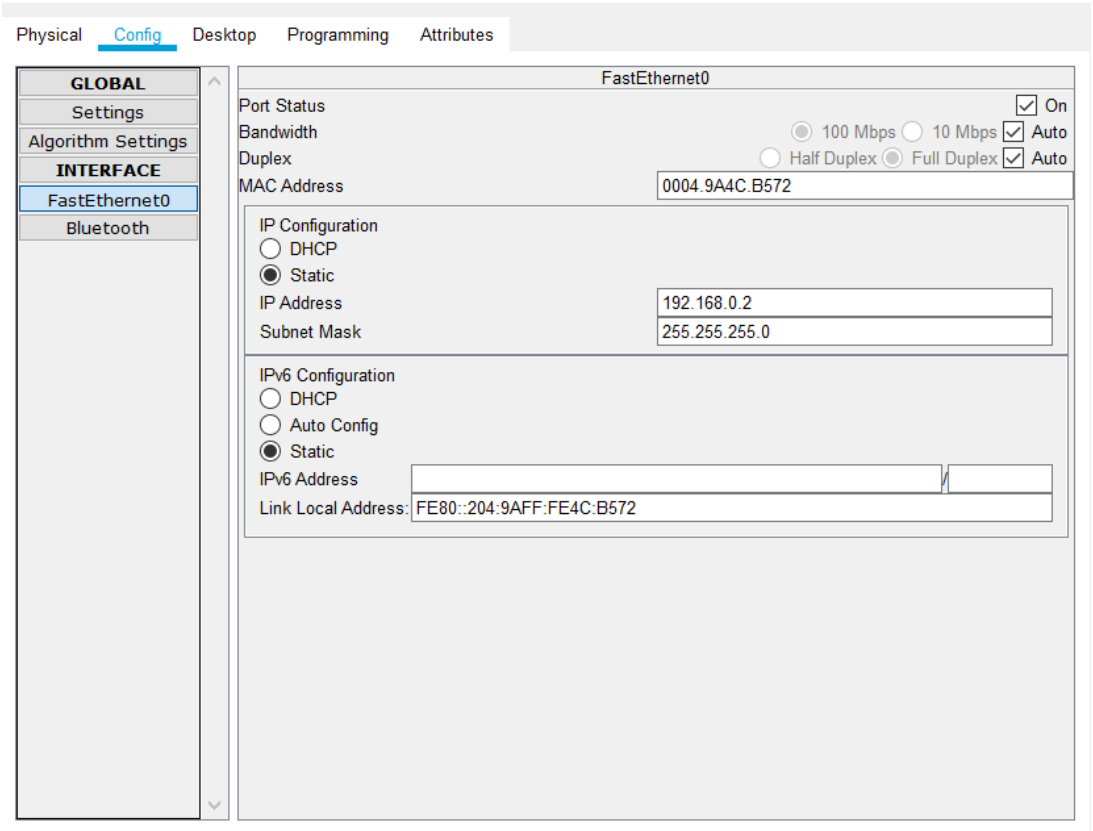
Expliqueu els diferents passos i com es relacionen els punts 1, 2, i 3 amb el punt 4.

Per començar hem creat la xarxa local demanada amb quatre ordinadors personals connectats entre si fent us d'un Switch, que com ha mínim ha de tenir aquestes quatre entrades (ports) per comunicar-se amb els PC's.
Com la xarxa es local no necessita cap connexió a "l'exterior" (internet) no necessitem de cap altre element.
Tots els ordinadors s'han connectat utilitzant el protocol Ethernet.



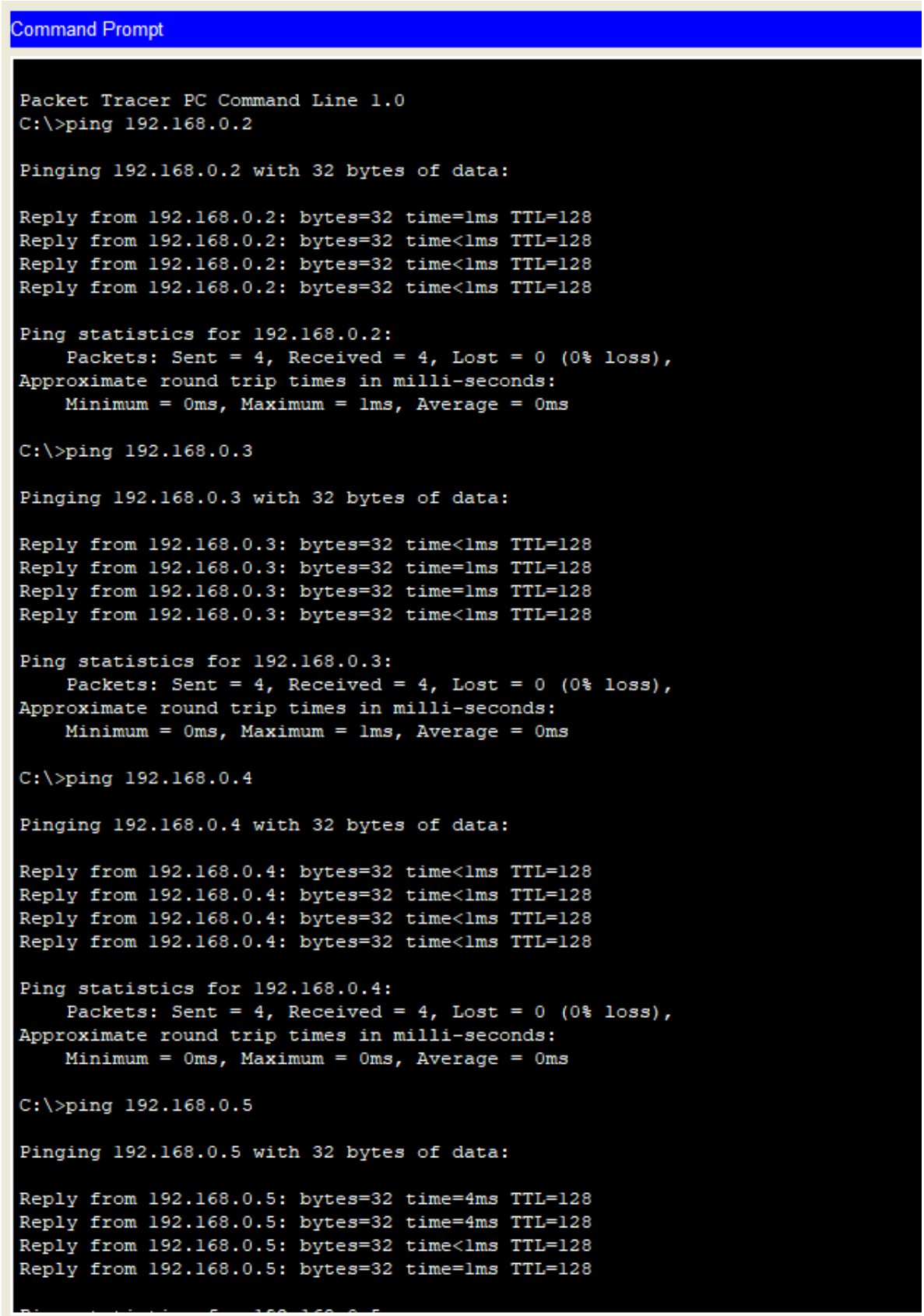
Imatge 3. Estructura de la LAN construïda.

Seguidament l'únic que ens queda es definir la IP estàtica de cada PC.
Les IP's configurades van de la 192.168.0.2 a la 192.168.0.5, sense utilitzar la 192.168.0.1 ja que normalment s'utilitza per "routers" o altres elements de la xarxa.
La màscara de la xarxa és la 255.255.255.0 (Classe C).



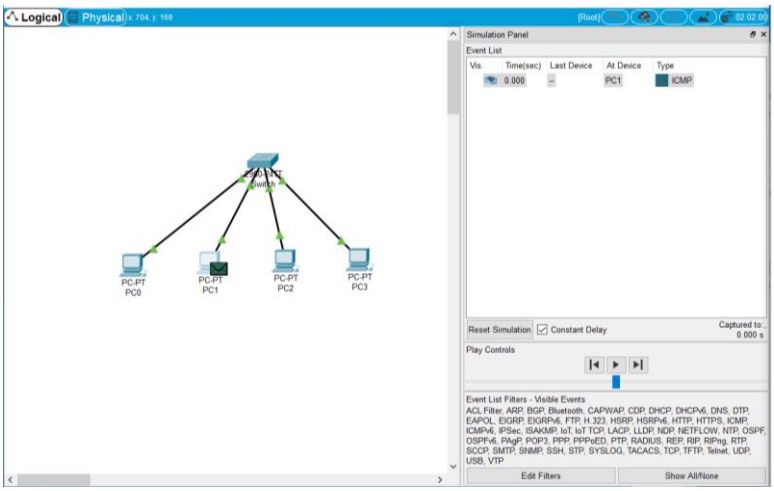
Imatge 4. Configuració de la IP dels PC's.

Només queda comprovar que la connexió entre PC's es bona, i per això podem provar a fer pings des de la terminal d'un ordinador a la IP d'un altre. Aquets son els resultats obtinguts.

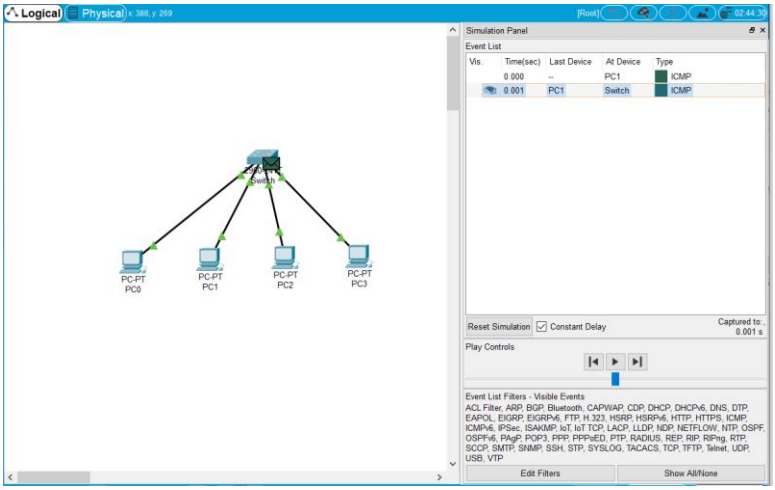


Imatge 5. "ping" des de PC0 a la resta dels PC's de la subxarxa.

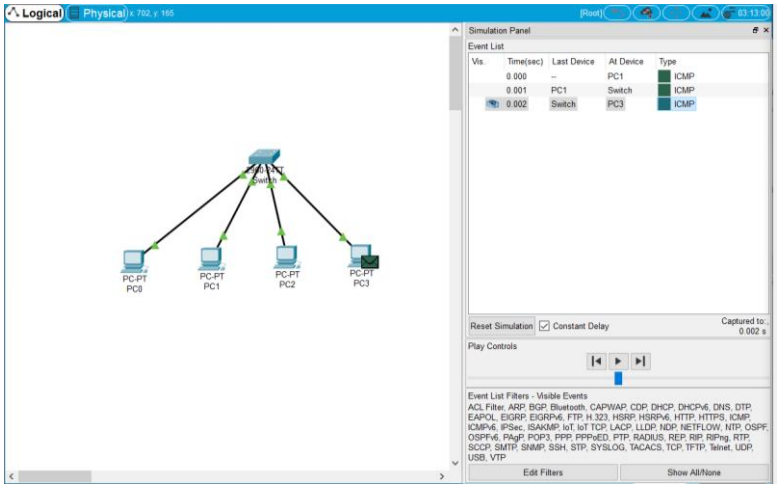
Com es pot veure tots els resultats han sigut correctes i podem dir que hi ha connexió entre tots els elements que formen el sistema. Hem observat el camí que segueixen els paquets en la comunicació entre dos PC's mentre s'executa l'operació.



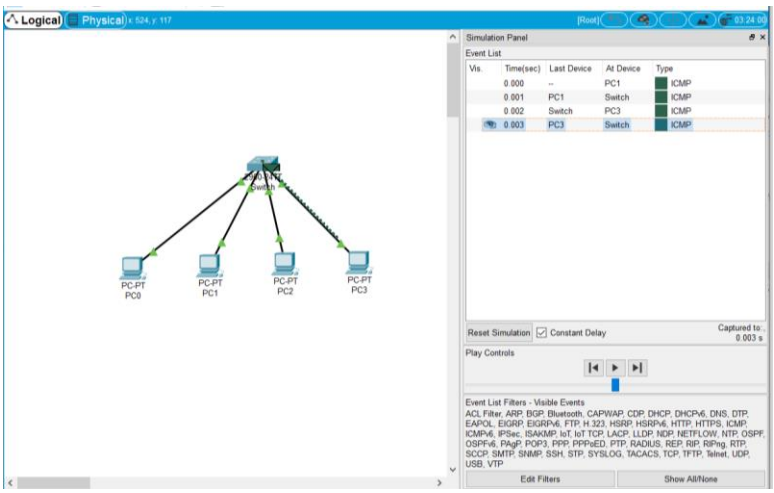
La ARP construeix la trama de demanda ("request") a l'IP objectiu, i l'encapsula en la PDU en un marc d'Ethernet. FastEthernet0 envia el paquet.



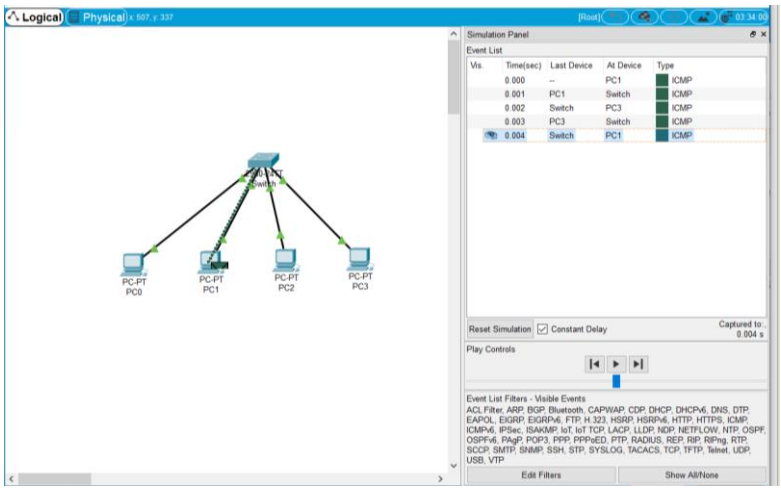
FastEthernet0/2 (al Switch) rep el paquet i com la font es troba a la taula MAC del Switch, el dispositiu desencapsula la PCU i processa les dades per conèixer el destí.
Pot passar que si no es coneix el destí la primera vegada que es fa ping s'enviï a traves de l'adreça de broadcast.
S'envia el paquet directament al destí o a tots els dispositius de la xarxa local (excepte el dispositiu origen), i s'espera a la resposta.



El dispositiu destí (PC3) rep el paquet a través de FastEthernet0 i comprova que la adreça MAC de destí coincideix amb la del port i desencapsula la PCU confirmant que la IP destí també coincideix amb la seva i llegeix el paquet ICMP que es un “Echo request”, forma la “Echo reply” i genera la trama i l’envia a la subxarxa.



El paquet arriba al Switch i busca la MAC de destí, i envia el paquet pel port d’accés del PC1.



El PC1 rep la resposta de PC3 i comprova que les MAC siguin les corresponents, desencapsula la PDU del “Ethernet Frame” i comprova que les IP siguin correctes, i processa el “Echo Reply” obtenint la informació demanada per la comanda ping.

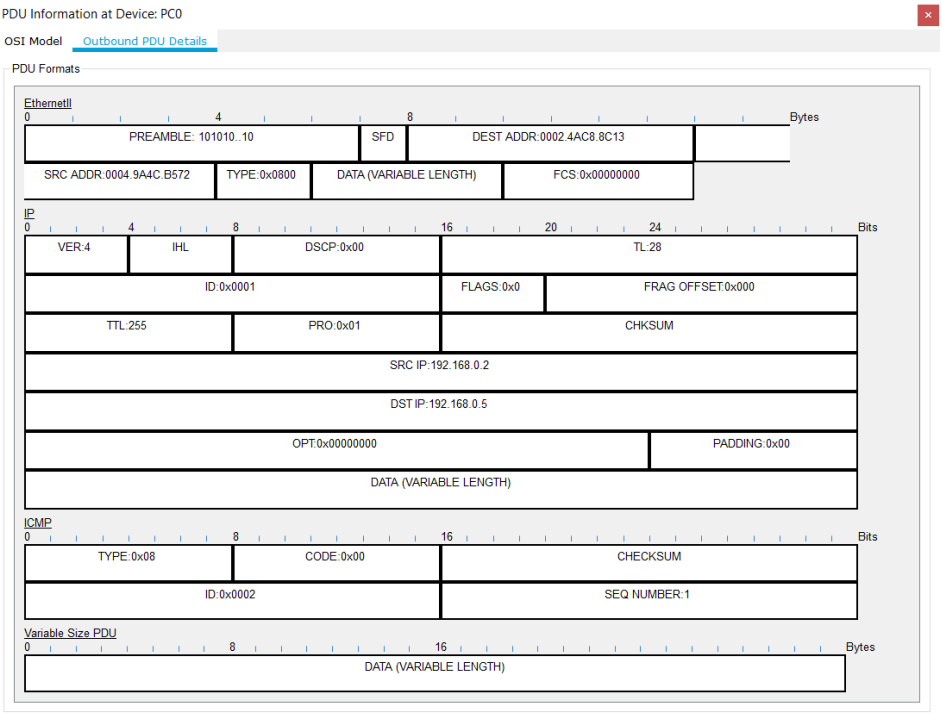
Event List

Vis.	Time(sec)	Last Device	At Device	Type
	0.000	--	PC1	ICMP
	0.001	PC1	Switch	ICMP
	0.002	Switch	PC3	ICMP
	0.003	PC3	Switch	ICMP
	0.004	Switch	PC1	ICMP
	1.005	--	PC1	ICMP
	1.006	PC1	Switch	ICMP
	1.007	Switch	PC3	ICMP
	1.008	PC3	Switch	ICMP
	1.009	Switch	PC1	ICMP
	1.945	--	Switch	STP
	1.946	Switch	PC0	STP
	1.946	Switch	PC1	STP
	1.946	Switch	PC2	STP
	1.946	Switch	PC3	STP

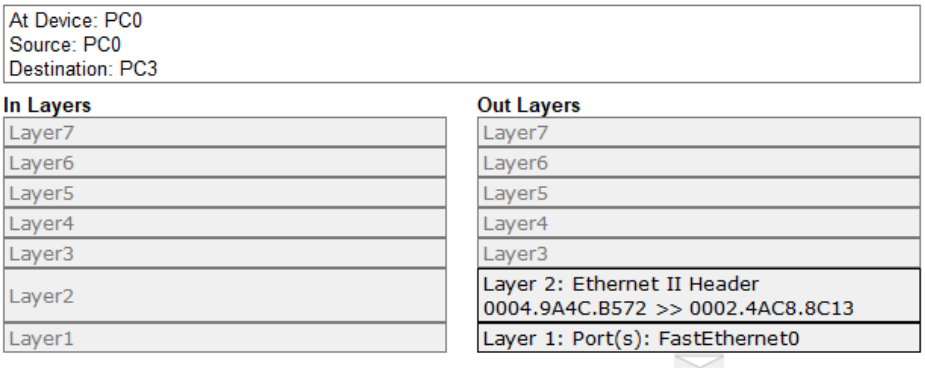
Imatge 6. “Event List” després de fer “ping” on es pot veure el paquet STP enviat per broadcast.

Com podem veure cada certa estona el Switch envia un paquet STP en “broadcast” a tots els equips connectats a la subxarxa. Es un paquet utilitzat pel “Spanning Tree Protocol”, de la capa 2 de OSI (capa de enllaç) que evita la presencia de bucles a tipologies de xarxes.

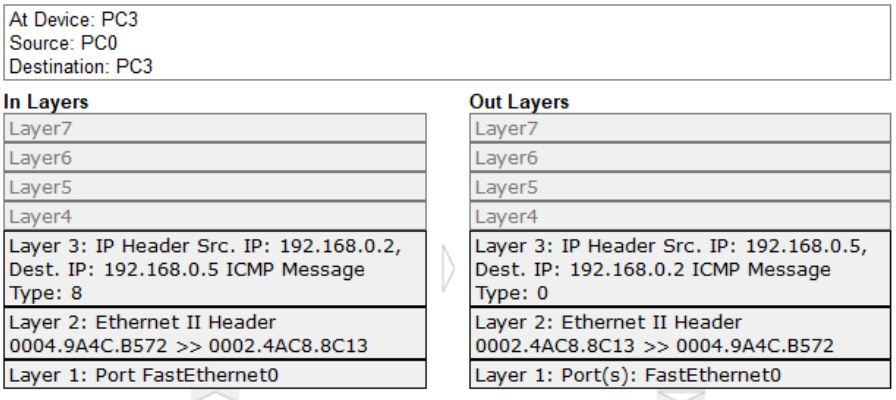
La trama es veu modificada pels diferents dispositius que per on passa, podem veure com canvia la trama original en el moment que arriba a l’objectiu, i la resposta que envia l’objectiu al l’origen.



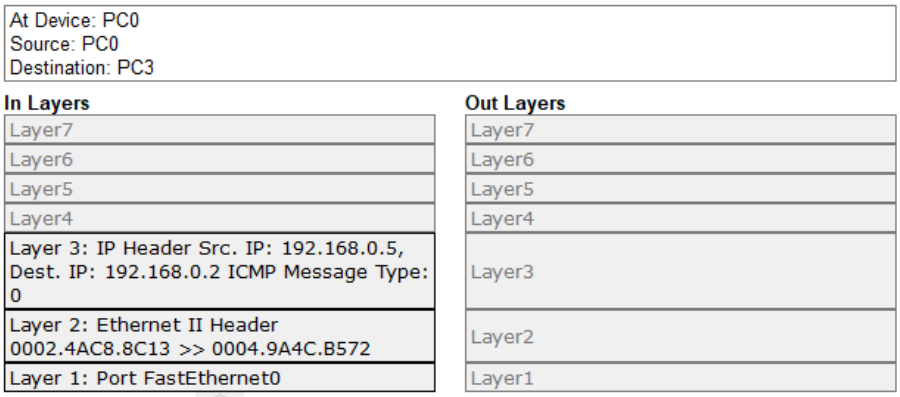
Imatge 7. Format de la trama Ethernet.



Imatge 8. Informació de la trama de sortida del PC0 fent ping a PC3.



Imatge 9. Informació de la trama de sortida del PC0 en arribar a PC3.



Imatge 9. Informació de la trama de "reply" del PC3 en arribar al PC.

CONCLUSIONS

En conclusió, creiem que hem assolit els objectius proposats inicialment ja que creiem que entès com funciona el protocol TCP/IP i hem sigut capaços de fer els programes demanades en esta pràctica.

En el primer programa, on havíem de crear el canal recol·lector de dades hem tingut un petit contratemps ja que quan enviavem les dades amb el String "body" ho fèiem amb client.println(). És a dir, que al final de la dada RSSI s'afegia un caràcter que simbolitzava final de línia. Per tant, encara que ThankSpeak rebia les dades no ens les imprimia en la gràfica. Per solucionar-ho hem hagut de revisar el codi amb precaució i entendre el seu funcionament, a més d'entendre el perquè d'aquest error.

En el segon programa, on havíem de crear la nostra pròpia xarxa WiFi amb la mota creiem que hem complit amb el que es demanava. El major problema amb el que ens hem trobat han sigut les funcions serializable i deserializable del JSON, per tal d'enviar i rebre les dades. El problema ha estat que el servidor no tenia delay i intentava llegir sempre abans de que el client acabés.

I finalment, l'últim apartat on simulem una red local de 4 ordinadors personals amb un Switch ha sigut bastant directa y hem aprés molt com es el camí que segueix un paquet i les modificacions que es fan, i quin es el procés quan es fa "ping" a un altre dispositiu, que es divideix en dues fases on s'envia el "request" al ordinador destí i aquest en la segona envia un "reply" amb les dades demanades a l'origen.