Catholic University of Louvain

Relay DHCP

LINGI2315 - Design of Embedded and Real Time Systems

Auteurs :
Vannoorenberghe Ludovic
(1045-10-00)
Vanwelde Romain (3143-10-00)

Groupe 7

Superviseurs :
Pr. Marc Lobelle
Emery Assogba

19 mai 2014

Table des matières

1 Introduction

Dans le cadre de ce travail, nous allons vous présentés les différentes étapes par lesquelles nous sommes passé pour la conception d'un relai DHCP amélioré.

Pour cela, nous avons d'abord crée celui-ci de manière conceptuelle, et grâce à un diagramme ASG. En partant de là, nous vous présenterons les limitations de notre environnement de programmation, la manière dont nous avons résolu nos problèmes, et nos seconde version du diagramme ASG.

Ce document se termine par une explication sur la manière dont vous pouvez utiliser notre code pour faire tourner, chez vous, un nouveau type de relai DHCP.

2 Fonctionnement du relai DHCP

Un simple relai DHCP se contente de transmettre les messages d'un client vers un serveurs, et d'un serveur vers un client. Lors du passage par le relai, deux champs du header sont quand même modifiés, le giaddr (à qui on attribue l'adresse su relai), et hops (à qui on incrémente la valeur de 1).

Dans le cadre de ce projet, il nous a fallu non seulement créer il relai DHCP, mais il surtout lui rajouter des fonctionnalité en plus.

1. Le relai n'offrira une adresse que pour une durée de 5 minutes ("lease time") mais mémorisera la durée de validité accordée par le serveur éloigné.
2. Il répondra lui même aux clients lorsque ceux-ci demanderont de renouveler leur adresse ("renew") sans consulter le serveur.

Pour pouvoir satisfaire ces deux conditions, nous devons stocker la liste des adresses déjà attribuées à des clients, ainsi que leur "lease time", et la durée de validité attribuée par le serveur. Lorsque le client effectuera une `DHCP_REQUEST`, on regarde d'abord dans la table si celui-ci est connu, et s'il possède déjà une addresse valide, dans quel cas, on retourne directement un message `DHCP_ACK` avec un lease time du minimum entre le temps de vie restant attribué par le serveur, et 5minutes. Par contre, si le client n'est pas encore connu, le relai transmet le message au serveur, le serveur repond avec un ACK, et cette fois, le relai va stoker l'adresse attribuée ainsi que sa période de validité et puis il va envoyer le message ACK pour le minimum entre 5minutes et le temps attribué par le serveur.

3. le relai ajoutera dans les informations transmises à l'hôte demandeur 'le client) l'adresse de diffusion (broadcast) du réseau local et l'adresse du routeur de sortie de ce dernier.

Lors du parcours des options à la lecture, on vérifie si les options citées précédemment sont déjà présentes ou non. Si elle sont présentes, on les modifie si besoin est, sinon, on les rajoutes.

4. Il demandera lui même au serveur un renouvellement lorsque la durée de validité accordée par le serveur approchera de son échéance.

Pour cette fonctionnalité, il faut se pencher sur la structure des adresses allouées avec le période de validité. Dès qu'une de ses durée passe en dessous de 10 secondes (durée arbitraire fixée par nous même), une message `DHCP_REQUEST` est automatiquement envoyé au serveur.

5. Si le client manque 5 échéances consécutives de demande de renouvellement d'adresse, il enverra un message "release" au serveur éloigné et effacera le client de sa liste.

Pour cette fonctionnalité, il nous est nécessaire de rajouter une variable pour chaque entrée de notre table (contenant les adresses et leur lease time) qui correspondra au nombre de demandes de renouvellement raté. Lorsque le compteur arrive à 5, le message correspondant est construit et envoyé au serveur.

3 Représentation sous ASG

Sur base de toutes les fonctionnalités listées ci-dessus, nous pouvons créer un diagramme ASG qui nous permettra d'avoir une meilleure vue de la manière dont on construira ensuite notre programme.

La figure ci-dessous montre notre premier diagramme ASG réalisé. Nous observons 4 composantes parallèles différentes. La première est chargée d'écouter sur les sockets, et d'attendre de nouveaux messages qu'elle stockerait ensuite sue un buffer, la seconde est chargée d'actualiser le temps ainsi que tout les lease time. La troisième s'occuperait de traiter tout les messages qui se trouvent sur le buffer, pour terminer avec la quatrième qui s'occupe de monitorer les validité des adresses attribuées (Renouveler l'adresse lorsqu'elle arrive presque à expiration, ainsi que la supprimer lorsqu'elle a raté 5 fois le renouvellement).

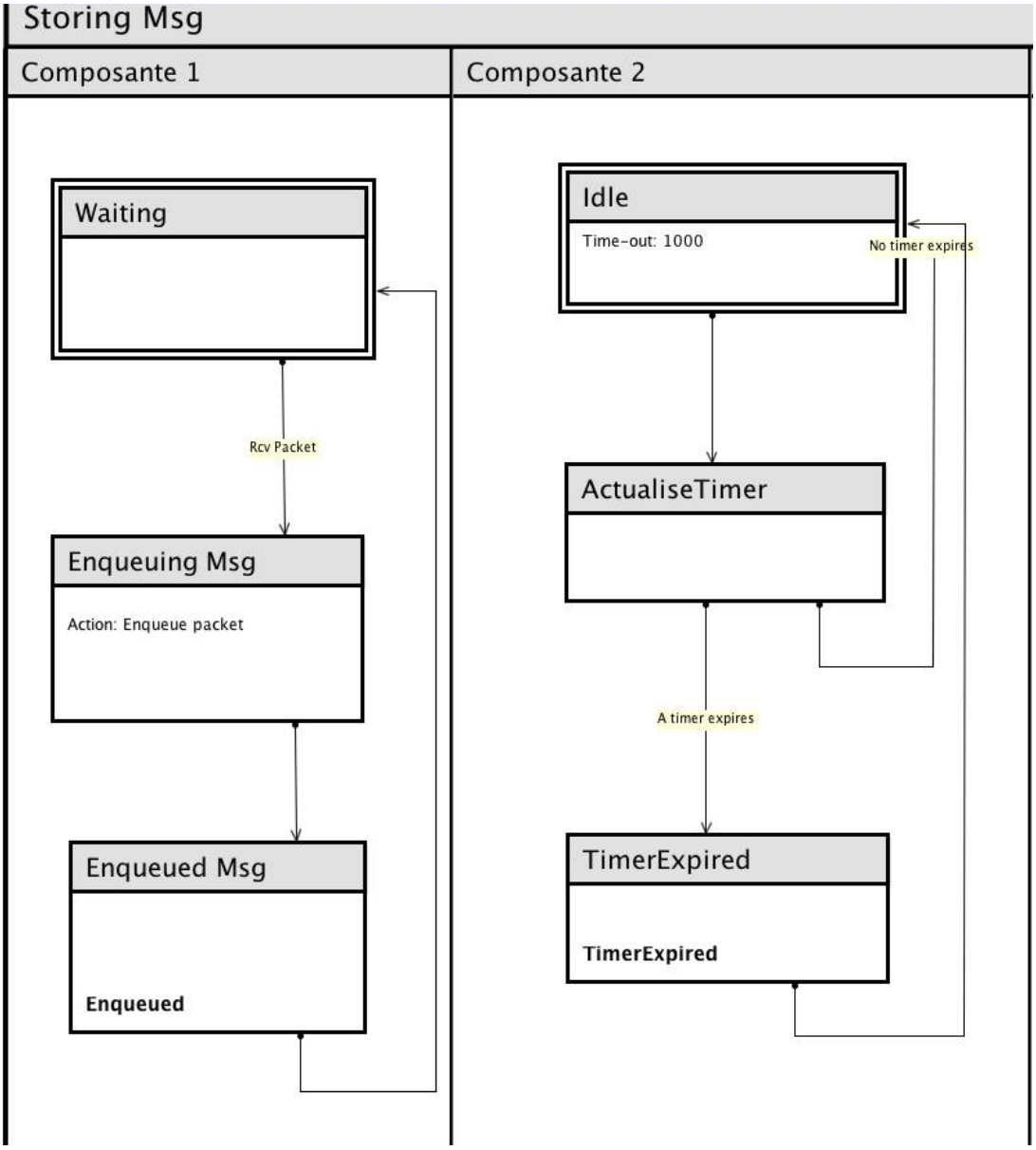

Figure 1 – Diagramme ASG V1 - Première Partie

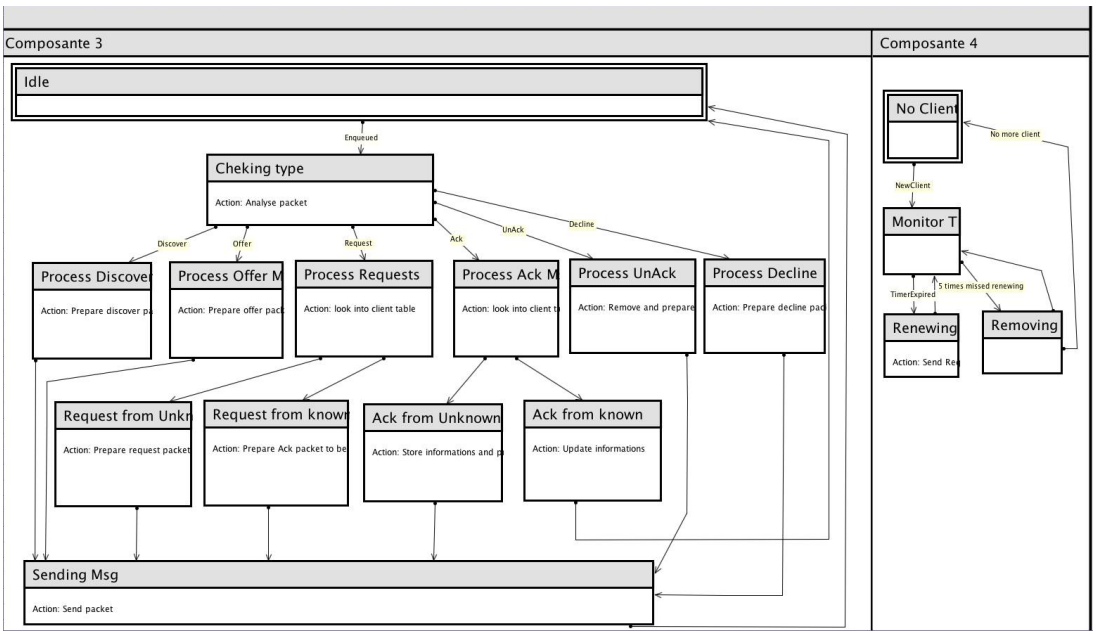

Figure 2 – Diagramme ASG V1 - Deuxième Partie

La composante 1 sera gérée par les fonctions UDP (UDPIsGetReady, ...) de TCP/IP. La composante 2 sera géré grâce à `TickUpdate();`. La compoosante 4 sera gérée par la fonction `TimerTask`, et la composante 3, elle sera finalement décomposée en deux composantes différentes, une première qui traite les messages du serveur, et une seconde qui traite les messages du client.

La plus grande différence entre notre diagramme ASG de base, et notre code final, est la suppression de la majorité des états présent dans la composante 3, remplacés par un traitement complet du message dans l'état "Checking Type". Les raisons de ce changement sont expliquées dans de plus ample détails dans la section suivante de notre rapport.

Pour finir, voici le diagramme ASG correspondant à la version finale de notre code.

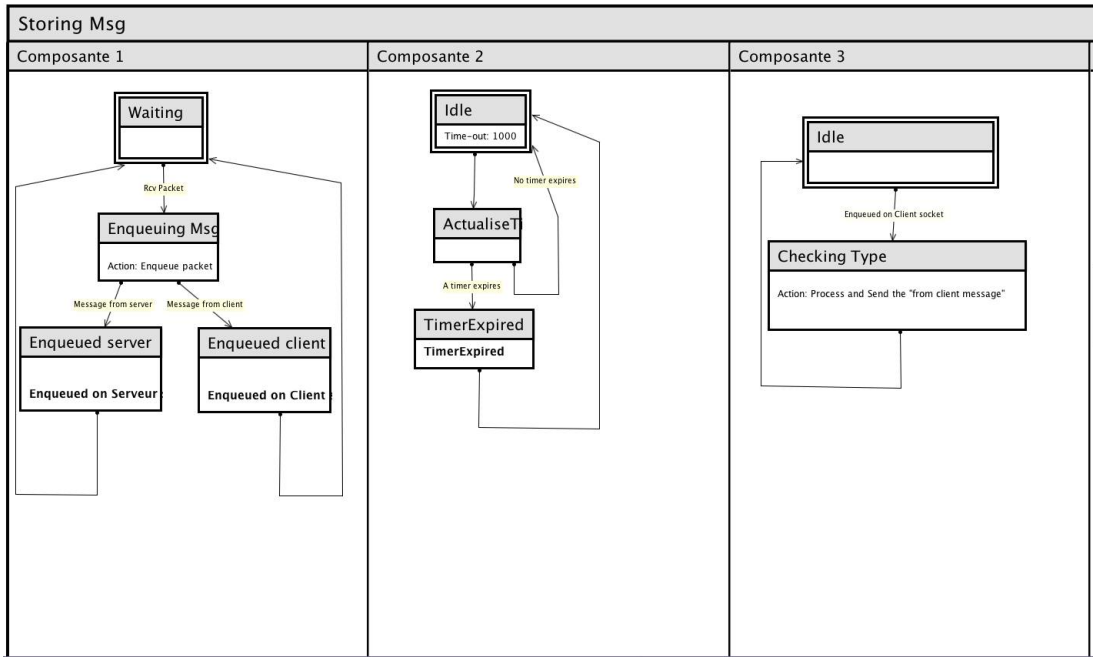

Figure 3 – Diagramme ASG V2 - Première Partie

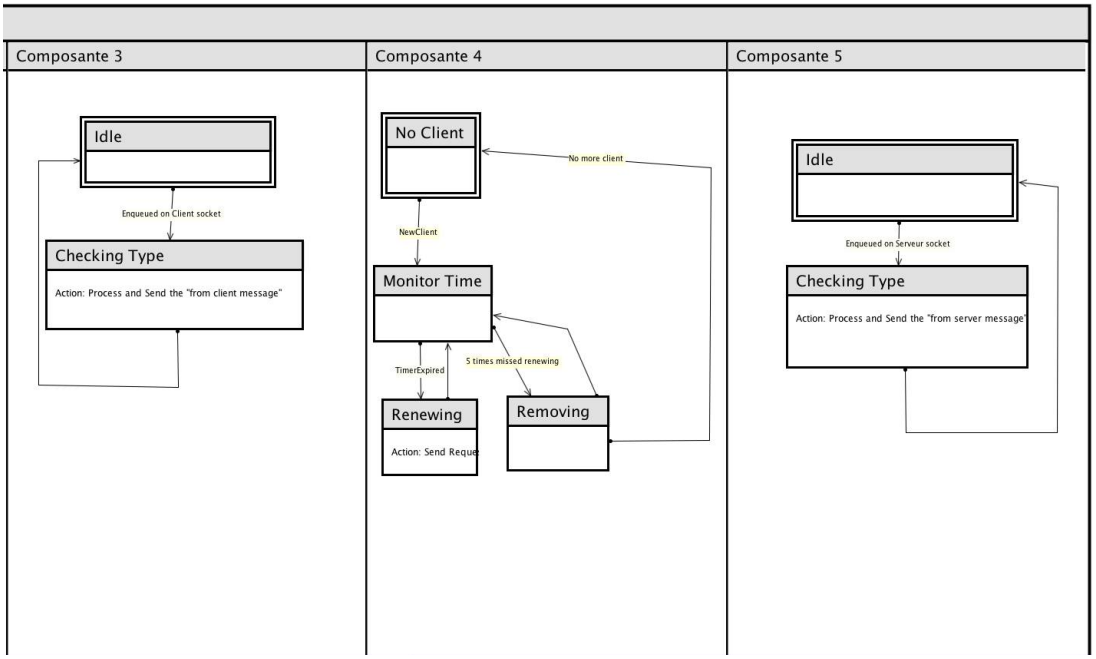

Figure 4 – Diagramme ASG V2 - Seconde Partie

On constate que la première composante a un état supplémentaire qui permet de distinguer l'ajout de messages sur le buffer du serveur/client. Ainsi que la composante 3 et 5 qui correspondent à l'ancienne composante 3 divisée en 2 pour traiter séparrement les messages provenant du serveur/client.

4 Limitations de l'environnement de programmation

Notre programme se base sur la pile TCP/IP de microchip et est conçu pour tourner sur un pic18. Cela amène quelques limitations dans la façon de structurer le programme.

Dans un premier temps, nous avions structurer notre programme selon le diagramme ASG que nous avions conçu. La première version de celui-ci possèdait un état différent pour chaque type de message. Ceci nous amenait à lire le message en entier afin de découvrir le type de celui-ci et ensuite décider de l'état suivant. En faisant cela nous nous sommes heurtés aux contraintes de la programmation sur machine nue.

4.1 Allocation dynamique

Premièrement, puisque nous lisions le message dans un état différent de celui dans lequel on le traite, nous devions le stocker dans un structure afin de toujours y avoir accès dans l'état suivant.

Nous avons donc créer une structure `DHCP_MESSAGE` contenant les différents champs que l'on retrouve dans un message dhcp, la première partie du message étant bien définie par le protocol, nous n'avons pas rencontrer de difficultés. Cependant, il n'y a aucun moyen de connaitre la taille du champ `DHCP` *options*. Pour solutionner ce problème, nous avons utiliser les instructions `calloc, realloc` et `free` pour allouer dynamiquement le champ *option*. Lors de la compilation, nous nous sommes rendu compte que ces instructions n'étaient pas implémentées sur des machines nues de type PIC et que nous devions les implémenter nous-mêmes si nous souhaitions les utiliser.

```
1 typedef struct
2 {
3     BOOTP_HEADER header;
4     BYTE mac_offset[10];
5     BYTE sname[64];
6     BYTE file[128];
7     BYTE magic_cookie[4];
8
9 } DHCP_MESSAGE ;
```

Nous avons adapté le structure de notre programme en conséquence ; nous récupérons le message et le traitons dans le même état et les options sont lues et écrites *à la volée* afin de ne pas avoir à stocker celles-ci.

4.2 Limitation de la taille des structures de données

Deuxièmement, le linker utilisé ne supporte pas les structures de données de plus de 256 bytes, nous avons donc du diviser nos structures en plus petites structures lorsque c'était nécessaire et enlever les variables de types `DHCP_MESSAGE` de notre état global (`DHCP_RELAY_VARS`) contenant certaines variables d'état indispensables.

```
typedef struct
{
    UDP_SOCKET client; // Handle to DHCP client socket
    UDP_SOCKET server; // Handle to DHCP server socket

    S2CSTATE s2cState;      // DHCP client state machine variable
    C2SSTATE c2sState;      // DHCP server state machine variable
    //Some variables for relay

    IP_ADDR my_ip;
    IP_ADDR router_ip;
    IP_ADDR broadcast_adress;

    NODE_INFO server_info;

    DHCP_CONTROL_BLOCK DCB[DHCP_MAX_LEASES];
} DHCP_RELAY_VARS;
```

5 Mise en place des tests

5.1 Environnement de test

Notre environnement de test est configuré selon la figure ci-dessous.

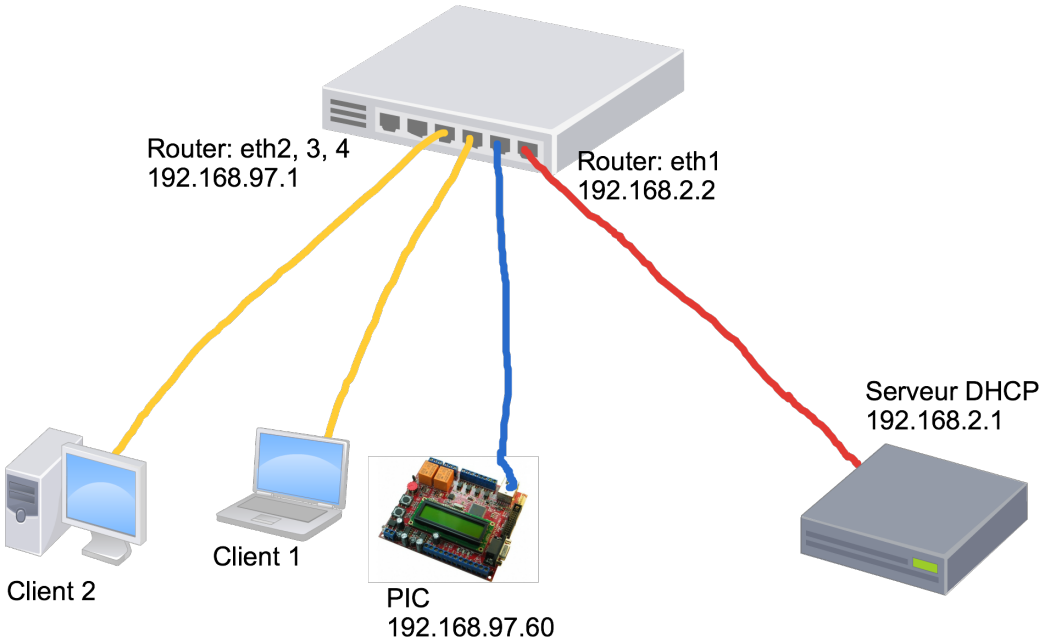

Figure 5 – Topologie du réseau

Voici les étapes que nous avons suivies pour configurer le routeur *MicroTik RouterBoard*

— Désactiver le DHCP Server du routeur sur toutes les interfaces
— Désactiver le DHCP Client du routeur sur l'interface eth1.
— Désactiver la fonction NAT du routeur.
— Attribuer l'adresse 192.168.97.1 sur les interfaces eth2, eth3, eth4, eth5.
— Attribuer l'adresse 192.168.2.1 sur l'interface eth1.
— Rediriger les packets provenant de l'interface 1, port 67 vers le pic.

5.2 Compilation et flash du pic

Pour compiler notre programme sur le PIC-MAXI-WEB il suffit de se rendre dans le dossier principal et de lancer la commande suivante :

```
make
```

Ensuite, connectez vous au routeur et assignez vous une adresse dans le range 192.168.97.2-192.168.97.254, lancez la commande tftp, activez les différents mode et terminez par la commande put :

```
tftp 192.168.97.60
tftp> binary
tftp> trace
    Packet tracing on.
tftp> verbose
    Verbose mode on.
tftp> put DHCPRelay.hex
```

Ne lancez la commande put qu'immédiatement après le reset de la carte Olimex !

5.3 Comment utiliser le relai DHCP

Notre relai DHCP se lance dès le démarrage et attend simplement de recevoir un message de la part d'un serveur. Celui-ci répondera automatiquement et affichera les différents message qui transite par lui sur son écran LCD. Il suffit donc de brancher le relai à l'alimentation et au routeur ayant pour adresse 192.168.97.1 et le relai fera son travail.

6 Conclusion

Ce projet nous a permis de découvrir, via ASG, les diagrammes à états hiérarchiques. Ceux-ci peuvent s'avérer utile pour bien structurer un programme qui s'apparente à une machine à états. Faire le diagramme avant de commencer l'implémentation nous a permis de réfléchir à certains aspects du système temp-réel auquel nous n'aurions pensé que tard dans le processus d'implémentation. Cependant, certains choix de design du système embarqué ainsi que l'absense de système d'exploitation nous ont contraints à revoir notre diagramme pour l'adapter à ce que nous avons implémenter dans la pratique. Nous avons ainsi pu observer les forces de la conception par diagramme ainsi que ses limitations.