amberMind Conseil et création technologiques

Cahier des charges

driver WIFI pour chipset Ralink RT2571W

sur hardware ARM7

RevA	13/03/2006	Création du document	Sylvain Huet
RevB	16/03/2006	Fusion des fonctions ARP et IP. Modification des milestones (échange scan/adhoc)	SH
RevC	18/04/2006	Mise à jour de l'API	Sébastien Bourdeauducq
RevD	25/05/2006	Mise à jour de l'API (Milestone 2)	SB
RevE	01/06/2006	Précision sur la taille de buffer de clef	SB

1 Introduction

1.1 Rappel du contexte

La société Violet édite et produit des objets communicants basés sur la technologie Wifi.

La société entre dans une nouvelle étape de son développement :

- développer un socle technique commun à plusieurs objets
- améliorer la scalabilité de sa plateforme
- améliorer la réactivité de ses objets

Il en résulte :

- une augmentation de la puissance de calcul des objets
  - passage à une technologie 32 bits
  - augmentation de la mémoire vive (3ko -> 1Mo)
- une augmentation de l'intelligence des objets
  - des objets qui maîtrisent leur connectivité réseau
  - des objets capables de définir leur comportement et de télécharger les ressources dont ils ont besoin
  - un bytecode plus complexe
- une meilleure gestion du réseau
  - une couche réseau qui ne recourt plus aux mécanismes de polling
  - des capacités de streaming in et out (dans le sens : l'objet joue la musique en même temps qu'il la télécharge)
  - une couche réseau qui est capable de gérer les indisponibilités de serveur

Pour atteindre ces objectifs, il est nécessaire d'utiliser une nouvelle interface wifi. Celle retenue est un dongle USB à base de chipset Ralink RT2571W.

L'USB est piloté par un driver USB maître, le ML60842 :
http://www.okisemi.com/datadocs/doc-eng/ml60842.pdf

Le hardware de l'hôte est le suivant :

- ARM7, 32Mhz, environ 30Mips
- 128ko de rom pour le programme
- 1Mo de SRAM

2 Périmètre du driver

2.1 Rappel des fonctionnalités réseau haut-niveau

L'objet est capable de :

- connaître son adresse MAC (celle en dur dans la carte wifi)
- découvrir les réseaux disponibles (scan)
- se connecter à une borne WIFI, en accès libre, en wep (open / shared), en wpa (PSK)
- effectuer des requêtes ARP
- obtenir une adresse IP à travers le protocole Dhcp (client)
- effectuer des requêtes DNS (client)
- envoyer/recevoir des paquets Udp
- créer une connexion Tcp
- effectuer une requête http

Dans la phase de configuration, l'objet est également capable

- de fonctionner en mode master (point d'accès)
- d'accepter les associations
- de répondre aux requêtes ARP
- de fournir une adresse IP à travers le protocole Dhcp (serveur)
- de recevoir une connexion Tcp
- de répondre à une requête http

2.2 Limite entre driver et application

L'implémentation des protocoles haut-niveau est à la charge de Violet : ARP, IP (et donc UDP, TCP, HTTP).

Le driver fournit une interface qui permet :

- de configurer la connexion wifi
- de connaître l'état de la connexion
- d'effectuer un scan
- d'envoyer des trames ARP ou IP
- de réceptionner des trames ARP ou IP

Le driver prend en charge :

- les échanges de trames liées au protocole wifi : connexion, authentification, échange de clefs, …
- le cryptage et le décryptage des données (WEP et WPA-PSK)
- le découpage et le réassemblage des trames qui le nécessitent

Le driver est écrit en langage C, sans utilisation de librairies.

L'utilisation de la sortie série pour les messages de debug est possible, mais doit pouvoir être désactivée et supprimée du code par un simple #define

3 API

L'application doit inclure le fichier rt2501usb.h uniquement.
Sauf mention contraire, les fonctions renvoyant un type « int » renvoient 1 en cas de succès, et 0 sinon.

3.1 Initialisation / configuration

int rt2501_driver_install(void);
Met en place le driver Ralink dans le driver USB Host.

void rt2501_timer(void);
Cette fonction doit être appellée toutes les 100ms par l'application, afin que le driver puisse recalibrer périodiquement la puissance d'émission de la carte et gérer les timeouts.
Cette fonction ne doit pas être appellée lorsque les interruptions sont désactivées.

extern unsigned char rt2501_mac[IEEE80211_ADDR_LEN];
L'adresse MAC de la carte se trouve dans ce buffer, sauf si le driver est dans l'état RT2501_S_BROKEN.

void rt2501_setmode(int mode, const char *ssid, unsigned char channel);
Définit le mode souhaité : IEEE80211_M_MANAGED ou IEEE80211_M_MASTER.
Dans le cas d'utilisation du mode Master, « ssid » et « channel » doivent donner les informations de configuration du point d'accès. Lors d'un passage en mode Managed, ces paramètres sont ignorés.
Il est possible de rappeller cette fonction lorsque l'on est en mode Master avec des paramètres « ssid » et « channel » différents, afin de reconfigurer le point d'accès.
Cette fonction ne doit pas être appellée lorsque les interruptions sont désactivées.

3.2 Scan

On définira une structure rt2501_scan_result, capable de contenir les paramètres de scan d'une borne :

```
struct rt2501_scan_result {
        char ssid[IEEE80211_SSID_MAXLEN+1];
        unsigned char mac[IEEE80211_ADDR_LEN];
        unsigned char bssid[IEEE80211_ADDR_LEN];
        short int rssi;
        unsigned char channel;
        unsigned short int rateset;
        unsigned char encryption;
};
```

puis on appellera la fonction suivante qui réalise un Scan et qui appelle la callback autant de fois qu'il y a de bornes à portée. Chaque appel de la callback fournit les informations d'une seule borne.

typedef void (*rt2501_scan_callback)(struct rt2501_scan_result *, void *);

void rt2501_scan(const char *ssid, rt2501_scan_callback callback, void *userparam);
Le paramètre « ssid » permet d'émettre, lors du scan, des « probe request » portant ce SSID. Ceci est nécessaire pour pouvoir se connecter sur des réseaux sans « SSID broadcast ».

Le paramètre « userparam » n'est pas interprété par la fonction mais est fourni à chaque appel de callback.
La fonction de callback peut être appellée plusieurs fois par le driver (à chaque réception d'un « beacon » ou d'une « probe response »).
Le callback est appellé depuis une routine de traitement d'interruption.

Le champ « rssi » donne la puissance du signal reçu, en dBm.

Le champ « encryption » peut prendre les valeurs 0 (IEEE80211_CRYPT_NONE) et 1 (IEEE80211_CRYPT_WEP64). Si le réseau utilise une clef WEP 128 bits, le champ sera tout de même égal à IEEE80211_CRYPT_WEP64. C'est à l'application de déterminer si le réseau utilise une clef 64 ou 128 bits.

Il n'est possible de scanner qu'en mode Managed.
Cette fonction ne doit pas être appellée lorsque les interruptions sont désactivées.

La connexion à un réseau s'effectue ensuite par l'appel de la fonction suivante :

void rt2501_auth(const char *ssid, const unsigned char *mac, const unsigned char *bssid, unsigned char channel, unsigned short int rateset, unsigned char authmode, unsigned char encryption, const char *key);

L'application doit ensuite appeler rt2501_state() afin de vérifier l'état de la connexion. En cas d'erreur de connexion, l'état passera à RT2501_S_IDLE ; en cas de succès, il passera à RT2501_S_CONNECTED.
Le paramètre « authmode » peut valoir 0 (IEEE80211_AUTH_OPEN) ou 1 (IEEE80211_AUTH_SHARED).
Le paramètre « encryption » prend les valeurs 0 (IEEE80211_CRYPT_NONE), 1 (IEEE80211_CRYPT_WEP64) et 2 (IEEE80211_CRYPT_WEP128). Ceci détermine la taille du buffer vers lequel pointe « key ». Si le cryptage n'est pas utilisé, le paramètre « key » est ignoré.
Si « authmode » vaut IEEE80211_AUTH_SHARED, « encryption » ne doit pas valoir IEEE80211_CRYPT_NONE.
Le buffer vers lequel pointe « key » doit contenir 8 octets pour une clef WEP64, et 16 octets pour une clef WEP128. Ses trois derniers octets sont ignorés, mais doivent être présents en mémoire pour des raisons d'alignement.
Cette fonction ne doit pas être appellée lorsque les interruptions sont désactivées.

3.3 Etat de la carte

int rt2501_state(void);
retourne l'état de la carte :

- 0 : (RT2501_S_BROKEN) « cassée » (carte absente ou ne répondant pas)
- 1 : (RT2501_S_IDLE) initialisée mais non connectée
- 2 : (RT2501_S_SCAN) scan en cours
- 3 : (RT2501_S_CONNECTING) connexion en cours d'établissement
- 4 : (RT2501_S_CONNECTED) connexion établie
- 5 : (RT2501_S_MASTER) mode Master

extern char ieee80211_assoc_ssid[IEEE80211_SSID_MAXLEN+1];

Si la carte est connectée, en cours de connexion, ou en mode Master, ce buffer contient le SSID sur lequel la carte se connecte, ou le SSID du réseau créé.

3.4 Envoi / Réception

Le driver est capable de mettre en attente une quantité arbitraire de trames reçues (tant qu'il y a de la mémoire disponible sur le matériel).

```
struct rt2501buffer {
        struct rt2501buffer *next;
        unsigned int length;
        char source_mac[IEEE80211_ADDR_LEN];
        char data[];
};
```

struct rt2501buffer *rt2501_receive(void);
Si une trame est en attente, celle ci est retirée de la file d'attente et renvoyée par cette fonction. Si aucune trame n'est en attente, la fonction renvoie NULL.
La structure rt2501buffer est composée des éléments suivants :

- next: utilisé par le driver lorsque la trame était encore en file d'attente, l'application peut ignorer ou altérer ce champ sans conséquence.
- length: taille de la trame
- source_mac: adresse MAC de l'émetteur de la trame
- data: données de la trame

La structure doit ensuite être libérée par l'application à l'aide de la fonction free().
La trame commence à l'en-tête LLC (Logical-Link Control), lequel est immédiatement suivi par une trame IP ou ARP (ce type est déterminé avec le LLC).

int rt2501_send(const char *frame, unsigned int length, const unsigned char *dest_mac, int lowrate, int mayblock);
Envoie une trame IP ou ARP. Celle ci doit être précédée d'un en-tête LLC.
Le paramètre « lowrate » permet, en mode Managed, d'envoyer la trame avec le débit le plus bas supporté par l'AP. Ceci est utile pour des trames courtes et critiques pour le fonctionnement du réseau (DHCP, ARP, ...).
Le paramètre « mayblock » indique à la fonction qu'en cas de buffer d'émission plein, elle peut attendre que de la place se libère pour y mettre la trame. Cette fonctionnalité ne doit pas être utilisée lorsque les interruptions sont désactivées.

4 Milestones

4.1 Phase A

- Réalisation du driver complet, sans cryptage, avec scan, et sans mode master

Le sous-traitant fournira également la description exhaustive et octet par octet des paquets émis et reçus.

4.2 Phase B

- Implémentation du driver WEP (64 et 128 bits : cryptage + mécanismes d'authentification)

- Implémentation du mode master

Le sous-traitant fournira également la description exhaustive et octet par octet des paquets émis et reçus.

4.3 Phase C

- Implémentation du driver WPA-PSK (cryptage + mécanismes d'authentification et de renouvellement des clefs)

Le sous-traitant fournira également la description exhaustive et octet par octet des paquets émis et reçus.