Département Informatique-Microélectronique-Automatique Polytech Lille

Rapport intermédiare de PFE

Partition HTTP/TLS pour Pépin

Mageshwaran SEKAR

sous tutelle de

Julien IGUCHI-CARTIGNY

IRCICA

Table des matières

Contexte

L'équipe 2XS (eXtra Small, eXtra Safe) de l'IRCICA en train de développer Pépin, un proto-noyau qui a une architecture plus simple (donc plus légère) qu'un noyau monolithique. Ce type d'architecture fonctionne plutôt dans l'espace utilisateur que l'espace noyau ce qui permet d'éliminer l'abstraction de matériel.

Cahier des charges

Dans cette section, je vais détailler l'objectif principal de ce projet et les différentes étapes à suivre afin d'atteindre ce but.

Objectif du projet

L'intérêt de ce projet est de développer une partition qui s'exécutent au-dessus de Pépin qui héberge un serveur web et implémente un protocole d'échange sécurisé de type TLS-PSK. Les clients doivent pouvoir connecter à ce serveur avec une clé qui a été échangée au préalable. Pour cela, il faut adapter un serveur web existant afin de pouvoir le porter au Pépin.

Choix techniques

Pour le développement de la partition de serveur web, on utilise le langage C. Pour tester le système, on va l'implémenter dans la carte embarquée Intel Galileo Gen. 2.

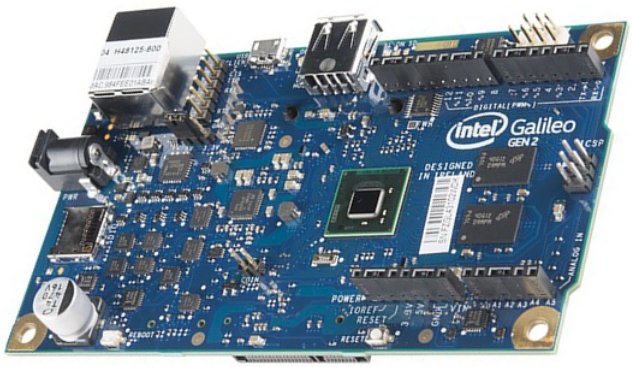


FIGURE 1 – Carte Intel Galileo Gen 2

Les étapes du projet

— Etudier l'architecture de Pépin
— Identifier une solution existante de serveur web qui peut être
— adapté pour le système
— Adapter le code du serveur web pour pouvoir utiliser avec Pépin
— Intégrer Pépin et le serveur web dans une carte embarqué et tester la solution

1 Etude de l'architecture de Pépin

Dans cette section, je vais détailler les différents types de noyaux et puis spécifiquement l'architecture de Pépin.

1.1 Noyau

Le noyau est une des parties fondamentales de systèmes. Il gère les ressources de l'ordinateur et permet aux différents composants — matériels et logiciels — de communiquer entre eux. Le noyau fournit des mécanismes d'abstraction du matériel, notamment de la mémoire, du processeur, et des échanges d'informations entre logiciels et périphériques matériels. Il autorise aussi diverses abstractions logicielles et facilite la communication entre les processus.

Le noyau est généralement exécuté dans un espace mémoire séparé de l'espace des applications : espace noyau, par opposition à l'espace utilisateur. Le passage entre ces deux espaces se fait via des appels systèmes. L'intérêt de cette séparation est que le système ne se plante pas si une des applications se plante.

1.2 Les différents types de noyau

Il existe plusieurs types de noyau (ex : monolithique, micronoyau, exonoyau, etc.) qui ont été développés pour des différents buts.

1.2.1 Noyau monolithique

Dans ce type de noyau, seules les parties fondamentales du système sont regroupées dans un bloc de code unique (monolithique). Les autres fonctions, comme les pilotes matériels, sont regroupées en différents modules qui peuvent être séparés tant du point de vue du code que du point de vue binaire.

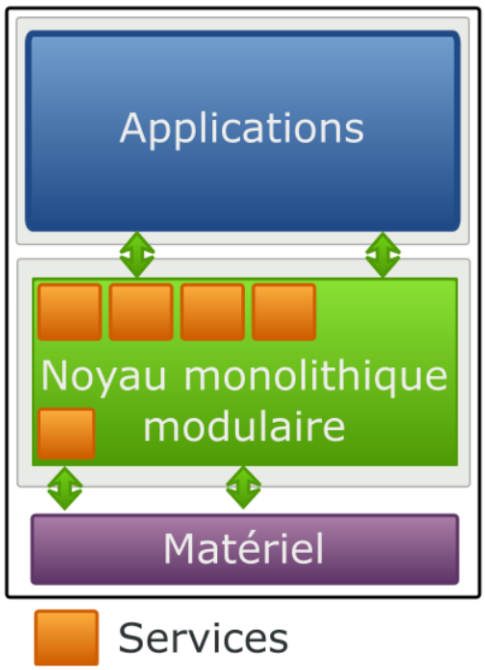


FIGURE 2 – Schéma de monolithique

La très grande majorité des systèmes actuels utilise cette technologie : Linux, la plupart des BSD ou Solaris. Par exemple avec le noyau Linux, certaines parties peuvent être non compilées ou compilées en tant que modules chargeables directement dans le noyau. La modularité du noyau permet le chargement à la demande de fonctionnalités et augmente les possibilités de configuration.

1.2.2 Micronoyau

Les systèmes à micro-noyaux cherchent à minimiser les fonctionnalités dépendantes du noyau en plaçant la plus grande partie des services du système d'exploitation à l'extérieur de ce noyau, c'est-à-dire dans l'espace utilisateur. Ces fonctionnalités sont alors fournies par de petits serveurs indépendants possédant souvent leur propre espace d'adressage.

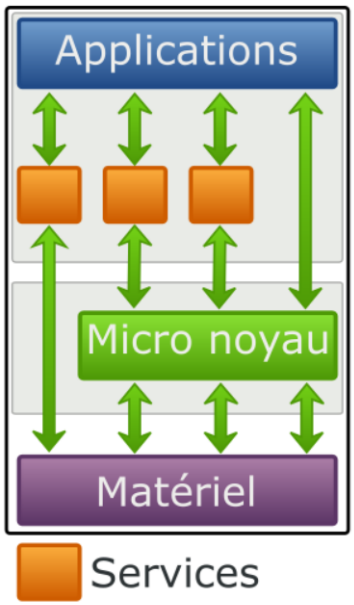


FIGURE 3 – Schéma de micronoyau

1.2.3 Exo-noyau

Un exo-noyau est un système fonctionnant dans l'espace utilisateur et non pas dans l'espace noyau. La philosophie de l'exo-noyau est l'élimination de toute abstraction afin que l'utilisateur soit au plus près du matériel.

De plus, le mécanisme de multiplexage des ressources permet à l'exo-noyau d'être souple et performant. En multiplexant les ressources du processeur, de la mémoire, du stockage et du réseau, plusieurs applications ont accès à celles-ci parallèlement et de manière équitable.

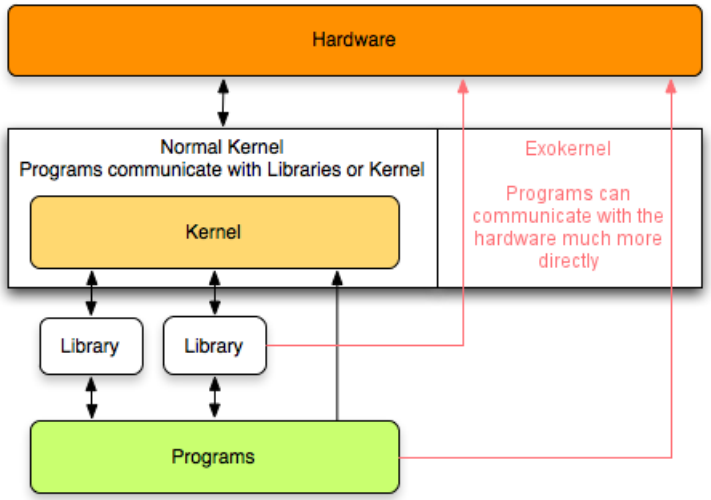


FIGURE 4 – Comparaison de noyau monolithique et exo-noyau

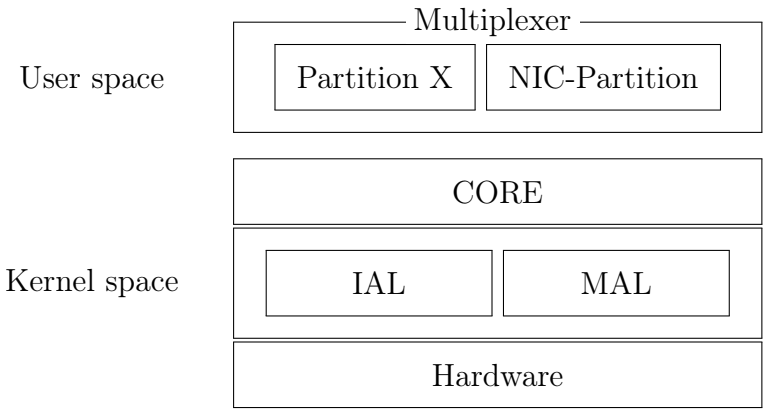
1.2.4 Protonoyau

Pépin est un proto-noyau qui est un dérivé d'un exo-noyau d'où le multiplexeur est situé au niveau de l'espace utilisateur. Son architecture est présenté en détails dans la partie suivante.

1.3 Pépin et son architecture

Pépin est un proto-noyau d'hypervision aux propriétés prouvable. Son but principal est d'assurer l'isolation mémoire entre plusieurs partitions (noyaux, applications self-standing) s'exécutant au dessus, ces dernières pouvant gérer leur espace d'adressage au travers de l'API fournie par l'hyperviseur.

Pépin assure que l'isolation est respectée en tout point de l'exécution quel que soient les opérations demandées au système. Afin d'assurer un portage simplifié sur plusieurs plateformes, la partie prouvable du code est réalisée en Coq, et est indépendante de l'architecture : elle contient toute la logique de sécurité que fournit le Pépin.



La gestion du matériel est fournie par le biais d'une couche d'abstraction du matériel (HAL) décomposée en deux parties : l'abstraction de la mémoire (MAL) et l'abstraction des mécanismes d'interruptions (IAL).

- **IAL** : sert à la gestion de l'intérruption (activer, désactiver, configurer, etc.)
- **MAL** : sert à la gestion de communication avec la mémoire (MMU)
- **BOOT** : contient le code pour démarrer Pépin
- **CORE** : contient le code principal de noyau de Pépin

2 Serveur web adapté pour Pépin

Dans cette section, je vais détailler le travail d'identification d'un serveur web existant qu'on va porter au Pépin. Je vais d'abord tester ce serveur sous Linux afin de faciliter le portage.

2.1 Fonctionnement d'un serveur web

Un serveur Web est un programme qui utilise le protocole HTTP pour fournir les fichiers qui constituent les pages web que les utilisateurs ont demandées via des requêtes transmises par les clients HTTP. Avoir un serveur web tout seul n'est pas souhaitable car on doit s'assurer que les données qui transitent entre le serveur et les client soit encrypté. C'est pour cette raison qu'il est essentiel d'utiliser la sécurisation par SSL/TLS.

2.2 La sécurisation avec SSL/TLS

Dans cette section, on va voir les différents types de chiffrement et choisir le mieux adapté pour un sytème embarqué.

2.2.1 Généralité de SSL/TLS

La technologie SSL/TLS (Secure Socket Layer) est utilisée pour sécuriser la transmission de données sur Internet : elle chiffre et protège les données transmises à l'aide du protocole HTTPS. La sécurité apportée par SSL/TLS repose sur 2 principes :

- **Le chiffrement des informations** : Toutes les données véhiculées sont rendues inintelligibles sauf entre le client établissant la connexion et le serveur sur lequel le site web se situe
- **L'authentification**(dans le cas asymmétrique) : Il permet l'authentification des échanges réalisés entre un client et le serveur, nécessaire à la sécurisation des transferts de données.

Il existe 2 grands types de chiffrements : asymétrique et symétrique aui sont utilisés actuellement.

2.2.2 Chiffrement Asymétrique

Les algorithmes de chiffrement asymétrique sont basés sur le partage entre les différents utilisateurs d'une clé publique ou l'utilisation d'un certificat signé qui est utilisé dans les pluparts de cas. La figure ci- dessous explique l'utilisation d'un certificat pour chiffrer les données entrer le serveur et le client HTTP.

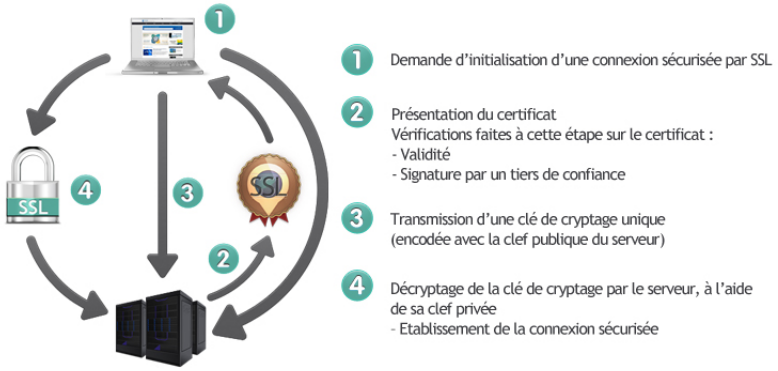


FIGURE 5 – Schéma d'établissement de connexion sécurisée

2.2.3 Chiffrement Symétrique

Parfois, dans les sytèmes embarquée la ressource de calcul est très limitée. De ce fait, on préfère d'utiliser un chiffrement symétrique qui a une mécanisme très différente de chiffrement asymétrique.

Le chiffrement symétrique se consiste à utiliser une valeur courte (la clé) pour rendre un message inintelligible aux tierces parties. Elle est dite symétrique car cette même clé permet ceux qui en ont connaissance de déchiffrer le message et ainsi d'accéder son contenu. Il valide la contrainte de confidentialité, aussi longtemps que la clé reste secrète.

Liste non-exhaustive de chiffrement par PSK (PreShared Key) :

- **TLS_PSK_WITH_AES_128_GCM_SHA256**
- **TLS_PSK_WITH_AES_256_GCM_SHA384**
- **TLS_DHE_PSK_WITH_AES_128_GCM_SHA256**
- **TLS_DHE_PSK_WITH_AES_256_GCM_SHA384**
- **TLS_RSA_PSK_WITH_AES_128_GCM_SHA256**
- **TLS_RSA_PSK_WITH_AES_256_GCM_SHA384**

Sachant que la sécurisation est importante pour un serveur web, il doit être considéré comme un élemént obligatoire lors de choix de ce serveur. Ce sera mieux si ce serveur supporte plusieurs types de chiffrement afin de laisser le choix lors de son implémentation.

2.3 Les différents serveur web existant

Dans cette section, on va discuter les différents serveur web et leurs avantages et inconvéninets afin de pouvoir choisir une solution de serveur web sécurisé pour être porté dans Pépin.

2.3.1 Apache

Apache HTTP Server (Apache) est un serveur HTTP créé et maintenu au sein de la fondation Apache. C'est le serveur HTTP le plus populaire et le plus répandu sur Internet. Apache est conçu pour prendre en charge de nombreux modules lui donnant des fonctionnalités supplémentaires : interprétation du langage Perl, PHP, Python et Ruby, serveur proxy, Common Gateway Interface, Server Side Includes, réécriture d'URL, négociation de contenu, protocoles de communication additionnels, etc. Néanmoins, il est à noter que l'existence de nombreux modules Apache complexifie la configuration du serveur web.

Au lieu de mettre en œuvre une architecture simple, Apache fournit une variété de modules multiprocesseurs (MPM), qui permettent à Apache de s'exécuter dans un processus hybride (processus et thread) ou pour mieux répondre aux exigences de chaque infrastructure particulière. Cette fonctionnalité limite son fonctionnement dans un grand système et il n'est pas adapté pour les sytèmes embarquées.

2.3.2 lighttpd

lighttpd est un serveur HTTP léger et rapide du fait qu'il a une plus petite empreinte mémoire que d'autres serveurs HTTP ainsi qu'une gestion intelligente de la charge CPU.

Même s'il est vu comme plus léger que les autres serveurs, il n'est pas non plus adapté avec notre système car il a besoin d'un système d'exploitation classique ou embarquée.

2.3.3 Mongoose

Mongoose est un serveur web dédié pour l'embarqué. Il est très petite en taille qui est aussi adapté pour notre système. Pour aider le development, il y a un API en C. Pourtant, l'implémentation de SSL/TLS n'est pas aussi complet. En effet, cette implémentation pourra causer de problème pour le mecanisme TLS-PSK qu'on a choisi.

2.3.4 SMEWS

SMEWS(Smart & Mobile Embedded Web Server) developpé par l'équipe 2xs est un serveur web pour l'internet des objets (IoT). Il a son propre pile TCP/IP dédie pour le serveur web. Puisqu'on a une pile dédie que pour le serveur HTTP, on ne peut pas l'utiliser pour d'autre service. De plus, il n'implémente pas de SSL/TLS.

2.3.5 Résumé des différents serveurs

Même si on a trouvé des serveurs web légers, ils n'ont pas satisfait nos conditions d'adaptabilité et sécurité. De ce fait, je devrais trouver une autre solution qui peut satisfaire toutes les critères que nous avons fixés au début. La solution qu'on a trouvé est d'implémenter une pile TCP/IP complète et un serveur web au dessus de cette pile.

2.4 Pile TCP/IP

La suite TCP/IP est l'ensemble des protocoles utilisés pour le transfert des données sur Internet. Elle est souvent appelée TCP/IP, d'après le nom de ses deux premiers protocoles : TCP (Transmission Control Protocol) et IP (Internet Protocol).

Couches hautes	Application
	Transport
	Réseau
Couches bases	Liaison de données
	Physique

TABLE 1 – Pile TCP/IP

Chaque couche résout un certain nombre de problèmes relatifs à la transmission de données, et fournit des services bien définis aux couches supérieures. Les couches hautes sont plus proches de l'utilisateur et gèrent des données plus abstraites, en utilisant les services des couches basses qui mettent en forme ces données afin qu'elles puissent être émises sur un médium physique.

2.4.1 Couches bases

Les couches bases de la pile TCP/IP contient la couche physique (qui s'occupe de transformer les bits d'information en signaux électrique ou optique) et la couche liaison de données(qui permet l'envoi et la réception de t trames entre deux équipements voisins). Normalement ces couches sont gérées par la couche matériel d'un système.

Dans Pépin, nous avons une partition dans l'espace utilisateur(NIC-Partition, voir section 1.3) qui fait l'abstraction de matériel. Les applications doivent être pouvoir communiquer avec cette partition si elles souhaitent envoyer des paquets vers la carte réseau et ensuite vers le réseau. Puisque ces couches bases sont déjà prise en charge par le système, on s'intéresse plutôt des couches hautes.

2.4.2 Couches hautes

La **couche réseau**, quant à elle, assure principalement, l'acheminement des paquets depuis une source vers une destination.

La **couche transport** peut résoudre des problèmes comme la fiabilité des échanges et assurer que les données arrivent dans l'ordre correct. Elle détermine aussi à quelle application chaque paquet de données doit être délivré(différecié par un numéro de port pour chaque application).

En cherchant les solutions qui peuvent fournir ces fonctionnements j'ai trouvé **picoTCP** qui s'occupe de ces deux couches. Mais, il manque la sécurisation des données par SSL/TLS d'où **wolfSSL** fournit cette solution.

Pour gérer les requêtes HTTP(dans **couche application**), il suffit de recupérer son entête et décoder les informations demandées. Le serveur web doit, à son tour, fournir une réponse HTTP au destinataire(le client).

2.4.3 La solution choisie

La table ci-dessous résume la solution que j'avais choisi pour résoudre le choix de pile TCP/IP.

Application	Serveur web
Transport	wolfSSL
	picoTCP
Réseau	

TABLE 2 – Pile TCP/IP avec picoTCP et wolfSSL

3 Tester l'intégration de picoTCP et wolfSSL

Avant de porter la solution dans pépin, je l'ai d'abord testé dans Linux. Le premièr test été d'implémenter picoTCP tout seul sans wolfSSL pour vérifier son fonctionnement. Au préalable de ces tests, il fallait configurer l'interface TUN/TAP sur Linux

3.1 Interface TUN/TAP

Pour utiliser picoTCP sous Linux, il faut créer un tunnel TUN/TAP afin que le système d'exploitation (Linux) ne modifie pas la trame ethernet. C'est parce que l'interface de carte réseau va enlever l'entete Ethernet et va passer le payload (normalement les paquets IPs) au système. D'où l'utilisation de TUN/TAP sera utile. Un dispositif TUN/TAP peut être vu comme une interface réseau qui communique avec un programme utilisateur (dispositif logiciel) au lieu d'une vraie carte matérielle (TUN pour mimer un périphérique point à point, TAP pour mimer un périphérique Ethernet).

La création d'une interface TAP doit se faire par l'intermédiaire d'un programme de l'espace utilisateur. En fonction des usages, plusieurs outils différents permettent de manipuler ces interfaces. On peut citer *tunctl* qui fait partie du paquet *uml-utilities*.

```
sudo tunctl #add tap0 interface
sudo ifconfig tap 10.0.0.1 #assign IP to tap0
```

Afin que la configuration de TUN/TAP reste inchangé, j'ai ajouté les lignes suivantes dans le fichier */etc/network/interfaces* :

```
iface tap0 inet manual
   pre-up ip tuntap add tap0 mode tap user root
   pre-up ip addr add 10.0.0.1/24 dev tap0
   up ip link set dev tap0 up
   post-up ip route add 10.0.0.1/32 dev tap0
   post-down ip link del dev tap0
```

3.2 picoTCP seul

Pour tester le fonctionnement de picoTCP, on peut utiliser le ping (ICMP). La réponse de ping est considérée comme le bon (ou mauvais) fonctionnement de picoTCP. Dans ce test, l'interface tap0 a l'IP de 10.0.0.1 et le côté picoTCP, on a 10.0.0.2 comme IP.

```
struct pico_ip4 ipaddr, netmask;
struct pico_device* dev;

dev = pico_tap_create("tap0");

//attribute ip and netmask to a new interface in tap0
pico_string_to_ipv4("10.0.0.2", &ipaddr.addr);
pico_string_to_ipv4("255.255.255.0", &netmask.addr);

pico_ipv4_link_add(dev, ipaddr, netmask);

//ping the ip address NUM_PING times
pico_icmp4_ping("10.0.0.1", NUM_PING, 1000, 10000, 64,
ping_cb);
```

3.3 Glue picoTCP et wolfSSL

Maintenant, il fallait implémenter la glue qui permet de faire fonctionner wolfSSL au-dessus de picoTCP et ensuite ajouter des fonctions pour traiter les paquets HTTPS. Ensuite, on utilise une fonction callback qui permet d'initialiser le serveur web avec TLS-PSK en précisant le type de chiffrage (ex : PSK-AES256-CBC-SHA). Il est aussi important de préciser la clé PSK qui sera utilisé côté client pour communiquer avec le serveur.

On peut tester la connexion et échange de données avec le serveur en utilisant l'utilitaire fourni avec *openssl* sous Linux :

```
openssl s_client -connect <server_ip>:<port> -psk <clé_psk>
```

4 Adaptation de code de serveur web pour Pépin

Les bibliothèques de picoTCP et wolfSSL sont fait pour être porté dans n'importe quel système mais il faut d'abord réécrire certaines fonctions car elles sont dépendantes du système Linux en ce moment.

4.1 Allocation mémoire

Quelques unes des fonctions principales à porter vers Pépin, ce sont ceux concernantes **allocation dynamique** de mémoire.

La plupart des fonctions ayant des besoins en mémoire dépendant de l'usage qu'on en fait, il est nécessaire de pouvoir, à des moments arbitraires de l'exécution, demander au système l'allocation de nouvelles zones de mémoire, et de pouvoir restituer au système ces zones (libérer la mémoire. C'est pour cette raison qu'on a besoin des fonctions comme :

- **malloc** : qui alloue une espace de mémoire (dans le tas) avec une taille demandée
- **zalloc** : rassemble à malloc mais acvec une étape supplémentaire d'initialiser l'espaces alloués avec zéro
- **realloc** : sert à redimensionner les espaces mémoire allouées
- **free** : libérer les espaces alloués dynamiquement

Travail restant

Il reste encore des fonctions à porter. Il faut avoir une fonction qui est capable de retourner la date depuis le démarrage de système. Cette fonction sera utilisée pour attendre un délai comme *sleep*.

Il faudra aussi réécrire ou trouver une autre solution pour la fonction *pow* (puissance) et *log* qui sont utilisées dans la méthode d'encryption de clé de Diffie-Hellman.

Il est aussi important de trouver une méthode pour génerer des nombres aléatoires ou pseudo-aleátoires utilisé par wolfSSL.

De plus, je devrais interfacer picoTCP avec les fonctions de driver de la carte réseau de Intel Galileo afin de tester la solution dans cette carte.

Conclusion

Pendant ces 4 mois de projet, je me suis familisié avec le système de Pépin qui n'était pas aussi facile a comprendre au début. Cependant, l'apprentissage de ce type de système ramène à une compréhension globale de fonctionnement d'un noyau. Il est intéressant de savoir comment fonctionnent les noyaux pour pouvoir développer les applications.

Ce projet m'a permis de comprendre la sécurisation de données avec SSL/ TLS. L'aspect sécurité est essentiel dans le monde connecté et avoir des connaissances dans ce domaine est indispensable pour un développeur.

Enfin, il y avait aussi l'occassion de revoir les fonctions fondamentales comme l'allocation mémoire ce qui m'a permis d'approfondir les connaissances de gestion mémoire.

Références

[1] Alternative Web Servers Compared : Lighttpd, Nginx, LiteSpeed and Zeus. http://royal.pingdom.com/2008/04/17/alternative-web-servers-compared-lighttpd-nginx-litespeed-and-zeus/.

[2] Dynamic Storage Allocation. http://www.cs.virginia.edu/~son/cs414.f05/lec11.slides.pdf.

[3] Exokernel. http://wiki.osdev.org/Exokernel.

[4] Fonction TUN/TAP du noyau Linux. https://www.inetdoc.net/guides/vm/vm.network.tun-tap.html.

[5] Memory Allocators 101. http://jamesgolick.com/2013/5/15/memory-allocators-101.html.

[6] picoTCP wiki. https://github.com/tass-belgium/picotcp/wiki.

[7] SMEWS : Smart & Mobile Embedded Web Server. http://www.cristal.univ-lille.fr/2XS/smews/.

[8] Testing HTTPS with openssl. https://blog.yimingliu.com/2008/02/04/testing-https-with-openssl/.

[9] The PIP Protokernel. http://pip.univ-lille1.fr/.

[10] wolfSSL Porting Guide. https://www.wolfssl.com/wolfSSL/Docs-wolfssl-porting-guide.html.