REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Ecole Nationale Polytechnique

Département d'Electronique

DOOFAS R&D

Mémoire de projet de fin d'études
pour l'obtention du diplôme d'ingénieur d'état en électronique

Étude et implémentation du protocole industriel PROFINET pour les I/O devices

YAHIA AISSA Ilyas

Présenté et soutenu publiquement le (12/07/2021)

Composition du jury :

Président	**M. TAGHI Mohamed Oussaid**	**MAA**	**ENP**
Promoteur	**M. LARFI Abdelmoumen**	**Ingénieur.**	**DOOFAS**
Promoteur	**M. LARBES Cherif**	**Prof.**	**ENP**
Examinatrice	**M. BENALIA Nour El-Houda**	**Dr.**	**ENP**

ENP 2021

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Ecole Nationale Polytechnique

Département d'Electronique

DOOFAS R&D

Mémoire de projet de fin d'études

pour l'obtention du diplôme d'ingénieur d'état en électronique

Étude et implémentation du protocole industriel PROFINET pour les I/O devices

YAHIA AISSA Ilyas

Présenté et soutenu publiquement le (12/07/2021)

Composition du jury :

Président	**M. TAGHI Mohamed Oussaid**	**MAA**	**ENP**
Promoteur	**M. LARFI Abdelmoumen**	**Ingénieur.**	**DOOFAS**
Promoteur	**M. LARBES Cherif**	**Prof.**	**ENP**
Examinatrice	**Mme. BENALIA Nour El-Houda**	**Dr.**	**ENP**

ENP 2021

Dédicace

Je dédie ce travail à ma merveilleuse mère, mon père et mon frère,

À la mémoire de mes deux grands-pères,

À ma famille,

À mes amis
Et à tous ceux qui m'ont soutenu dans mon parcours.

Remerciements

Je voudrais dans un premier temps remercier, mon encadreur **M.LARBES Cherif**, pour sa confiance, il m'a toujours soutenu dans mes projets et a toujours été à l'écoute.

Je remercie également **M. LARFI Abdelmoumen** et **M. REBIKA Samir** ainsi que toute l'équipe de **DOOFAS** pour leur disponibilité, leurs précieux conseils et pour m'avoir introduit au monde professionnel.

Je tiens à témoigner toute ma reconnaissance envers mes parents pour m'avoir offert l'environnement adéquat afin de me focaliser sur mon travail, ainsi que mon frère qui m'a apporté son soutien moral et intellectuel tout au long de mon projet.

Je remercie sincèrement les membres du jury, **Mme. BENALIA Nour El-Houda** et **M.TAGHI Mohamed Oussaid**, qui ont généreusement offert leur temps, leur soutien, leurs conseils et leur bonne volonté durant ces trois dernières années.

Enfin, je remercie mes amis Adel, Zaki, Walid, Nadir, Djamel, Larbi, Nassim et Abderrahmane ainsi que mes cousins Arysse, Adlane, Anis, Nassim, Sofiane et Malik qui ont toujours été là pour moi. Leur soutien inconditionnel et leurs encouragements ont été d'une grande aide.

ملخص :

البروتوكولات الصناعية القياسية في الأنظمة الآلية عديدة اليوم .يمكن تصنيفها إلى فئتين :البروتوكولات القائمة على إيثرنت وغيرها .يتوافق كل بروتوكول مع حاجة وفقًا للأداء المتوقع والسوق المستهدف ، مع فكرة مركزية : الوقت الفعلي

تقدم هذه الوثيقة دراسة بالإضافة إلى تنفيذ اتصال جهاز الإدخال والإخراج باستخدام بروتوكول بروفينت على نظام التشغيل لينكس للدمج ، ثم التكيف من نظام التشغيل بحيث يلبي قيود الوقت الحقيقي .أخيرًا ، يقدم الاختبارات التي تم إجراؤها على جهاز الإدخال والإخراج والتي تؤكد الأداء السليم لنظامنا وكذلك احترام قيود الحتمية

الكلمات مفتاحية : بروفينت ، إيثرنت صناعي ، لينوكس مدمج ، الوقت الحقيقي

Abstract

Nowadays, there are many standard industrial protocols in automated systems. They can be divided into two categories : Ethernet-based and non-Ethernet-based protocols. Each protocol corresponds to a need according to the expected performances and the targeted market, with a central notion : real time.

This document presents a study as well as the implementation of an IO device communicating with the protocol PROFINET on embedded linux, then the adaptation of the operating system so that it meets the real time constraints. Finally, it presents the tests carried out on the IO device which confirms the proper functioning of our system as well as the respect of the constraints of determinism.

Key words : PROFINET, Industrial Ethernet, Embedded linux, Real time

Résumé

Les protocoles industriels standards dans les systèmes automatisés sont de nos jours nombreux. On peut les classer en deux catégories : les protocoles basés sur Ethernet et les autres. Chaque protocole correspond à un besoin selon les performances attendues et le marché visé, avec une notion centrale : le temps réel.

Ce document présente une étude ainsi que l'implémentation d'un IO device communiquant à l'aide du protocole PROFINET sur linux pour embarqué, Puis l'adaptation du système d'exploitation afin qu'il réponde aux contraintes de temps réel. Enfin, il présente les tests effectués sur l'IO device qui confirme le bon fonctionnement de notre système ainsi que le respect des contraintes de déterminisme.

Mots clés : PROFINET, Ethernet industriel, Linux embarqué, Temps réel

Table des matières

Liste des tableaux

Table des figures

Liste des abréviations

IO	Input/Output
SNCC	Système numérique de contrôle-commande
PLC	Programmable Logic Controller
GSD	General Station Description
I&M	Identification Maintenance
CIM	Computer-Integrated Manufacturing
CC	Conformance Classes
OSI	Open Systems Interconnection
UDP	User Datagram Protocol
IP	Internet Protocol
RTC	Real Time Class
VLAN	Virtual Local Area Network
MAC	Media Access Control
AR	Application Relation
CR	Communication Relation
IOCS	Input/Output Consumer Status
IOPS	Input/Output Provider Status
CM	Context Management
LLDP	Link Layer Discovery Protocol
DCP	Discovery and Configuration Protocol
TCP	Transmission Control Protocol
XML	Extensible Markup Language

Introduction

Au cours des dernières années, la transition vers le numérique a été la priorité des services publics, entreprises et industries, les efforts sont justifiables au vu des avantages qu'offre la digitalisation de nos informations. Il a été donc nécessaire d'adapter les équipements et machines afin d'assurer la possibilité de traitement de données numériques et de communication à l'aide de divers protocoles qui nous permettent le transfert et l'acquisition d'informations.

Cette philosophie du passage vers le numérique a rapidement été adoptée par les industriels, d'où le partitionnement des différentes périodes de l'industrie, nous sommes actuellement dans l'ère de l'industrie 4.0 qui est caractérisée par l'utilisation d'outils numériques qui permettent l'optimisation et le contrôle des processus industriels.

Parmi les entreprises qui ont mené ces changements, Siemens, ABB, allen and bradley ont joué un rôle important dans la conception et la commercialisation d'équipements d'automation industrielle. La diversité des automates disponibles a permis la création de différents protocoles de communication, chacun propre à une certaine gamme de produits ou marques. Ces protocoles de communication permettent aux dispositifs électroniques, représentés sous forme de système embarqué, dans un environnement industriel où ils contrôlent des machines et capteurs, d'être interconnectés dans un réseau permettant la communication entre les différents périphériques.

L'un des fournisseurs d'automates les plus influents SIEMENS a profité du développement de l'un des protocoles les plus utilisées, ethernet en l'occurence , pour concevoir le protocole PROFINET qui est basé sur cette technologie. Ce genre de protocole sera nommé ethernet industriel au vu de son utilisation dans un environnement industriel avec des contraintes de temps réel et de déterminisme.

Le but de ce travail est la conception d'un IO device sur système embarqué communiquant à l'aide du protocole de communication PROFINET.

Ce document est organisé comme suit :

- **Chapitre 1 (Aperçu global sur PROFINET)**

Dans ce chapitre, on introduit les concepts de base du protocole PROFINET en présentant les notions fondamentales y afférant ainsi qu'une description de l'environnement où il est utilisé.

- **Chapitre 2 (Protocole de communication PROFINET)**

Dans ce chapitre, le protocole de communication PROFINET est abordé avec plus de détails relatifs à ses caractéristiques et ses fonctionnalités.

- **Chapitre 3 (Implémentation du protocole PROFINET)**

Dans ce chapitre, on aborde les choix matériels considérés ainsi que les étapes nécessaires à l'implémentation de la librairie p-net pour le développement d'une application pour un cas pratique en respectant les contraintes de temps réel du protocole PROFINET .

- **Chapitre 4 (Tests et résultats)**

Dans ce chapitre, nous présentons les tests effectués sur notre système après implémentation puis nous procédons à l'analyse des résultats obtenus.

- **Conclusion générale**

Dans la conclusion générale, on énumère les principaux résultats de notre travail et on présente des perspectives.

Chapitre 1

Aperçu global sur PROFINET

1.1 Introduction

Dans l'optique de transition vers le numérique, les systèmes de contrôles et de commandes n'ont cessé d'évoluer, entrainant avec eux le développement de concepts et d'outils qui ont contribué à la numérisation de l'environnement industriel.

Cet environnement a engendré l'émergence de nouvelles technologies, parmi elles le développement de protocoles de communication afin d'assurer la communication entre les différents équipements industriels. L'un des protocoles qui a été développé par SIEMENS et qui s'est caractérisé par son utilité dans des applications à contraintes de temps réel et ses riches fonctionnalités, est le protocole PROFINET.

Ce chapitre a pour but d'introduire les notions de base concernant le protocole PROFINET ainsi que les concepts associés.

1.2 Définition

L'automatisation des procédés industriels a été une priorité pour les entreprises, elle a permis l'accroissement de la productivité, en plus de l'amélioration de la flexibilité et la qualité des produits.

Une installation industrielle typique dispose de capteurs et actionneurs qui seront contrôlés à l'aide de PLC et d'interface Humain-Machine, on peut alors parler de système numérique de contrôle-commande (SNCC), les équipements de commande d'un SNCC sont distribués ou géo-répartis, et disposent de protocoles de communication qui assure l'intéraction entre les différents équipements du système.

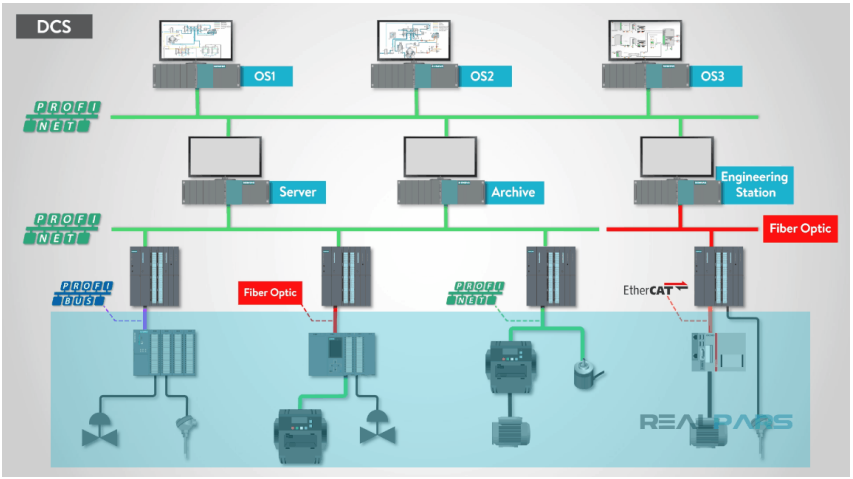

FIGURE 1.1 – Exemple d'un système numérique de contrôle-commande

1.2.1 Système numérique de contrôle-commande :

Un système de contrôle-commande permet le pilotage d'un procédé industriel physique au travers des fonctions de commande, de surveillance et de supervision (figure 1.2). La commande a un rôle opérationnel qui consiste à faire exécuter un ensemble d'opérations pour agir sur le procédé physique [1].

La surveillance a un rôle informationnel qui porte sur le recueil des signaux provenant du procédé, de la commande et le traitement des défaillances. La supervision a un rôle décisionnel qui permet d'optimiser, en présence ou non de défaillances, le fonctionnement du système. L'interaction du système de contrôle-commande avec le procédé physique est réalisée, d'une part, par des observations à travers des capteurs, et d'autre part, par des actions réalisées par l'intermédiaire d'actionneurs (figure 1.2). Les capteurs permettent de transformer les grandeurs physiques du procédé matériel en signaux électriques interprétables par le système de contrôle-commande. Les actionneurs (moteurs, transformateurs...) permettent de transformer les commandes électriques du système de contrôle-commande, en ordres qui permettent d'agir sur le procédé physique en changeant son état[2].

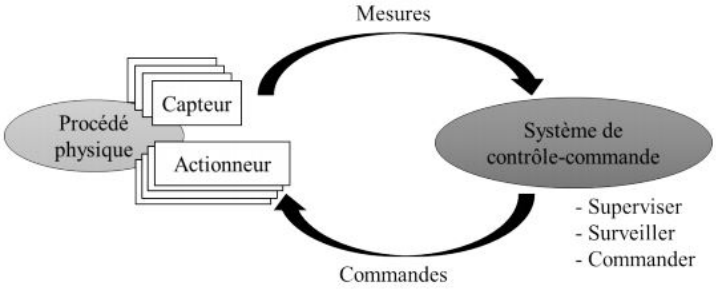

FIGURE 1.2 – Représentation d'un système de contrôle-commande

1.2.2 Programmable logic controller (PLC)

Un automate programmable (PLC) est un type particulier d'ordinateur. Même dans les environnements industriels difficiles, les automates sont incroyablement fiables. Un automate programmable est un contrôleur polyvalent qui utilise des données provenant d'un ensemble d'entrées, les traite par le biais d'une série de logiques préprogrammées et envoie une sortie physique en fonction des résultats de cette logique. Par conséquent, ces contrôleurs peuvent automatiser le fonctionnement d'une machine, d'un processus ou d'une chaîne de fabrication entière.

Il fonctionne comme le "cerveau" d'un système de contrôle, et peut donc interpréter les données lues en input et décider de ce qui doit se passer une fois qu'il a été correctement programmé. En fonction de cette décision, il manipule ensuite les signaux de sortie. Les équipements situés sur le terrain sont connectés à ces signaux de sortie. Il peut s'agir par exemple d'un interrupteur de démarrage/arrêt ou d'un IO device.

FIGURE 1.3 – Automate programmable compact PLC SIEMENS S7-1200

1.2.3 IO device (Appareil de terrain)

Ces dispositifs intègrent l'intelligence requise pour les techniques de détection et de contrôle simples . Ils remplacent donc le besoin d'un contrôleur pour effectuer les tâches basiques habituellement réalisées par un PLC.

Ils peuvent être directement connectés à un bus de terrain, ce qui permet de transmettre de multiples mesures à la station de contrôle de niveau supérieur suivante via une ligne de transmission numérique, en éliminant le matériel superflu tel que les modules locaux connectés au PLC.

Ces IO devices décentralisés relient des capteurs et des actionneurs binaires et analogiques au contrôleur mobile. Ils permettent l'évaluation décentralisée des signaux des capteurs et la commande d'actionneurs ou de vannes proportionnelles. La sortie des données et le réglage des fonctions de l'appareil se font via un bus d'interface.

FIGURE 1.4 – Débitmètre massique avec ethernet industriel (PROFINET)

1.3 Bus de terrain

Le bus de terrain est un système de réseau industriel pour le contrôle distribué en temps réel. C'est un réseau de communication numérique reliant différents types d'équipements d'automatisme intelligents ou à intelligence limitée pour permettre leur coopération tel que : les capteurs, les actionneurs, les automates programmables, les machines à commande numérique, les robots ...etc. Dans les réseaux de terrain, la taille des messages échangés est assez faible comparativement aux autres types de réseaux, locaux ou à grandes distances. Les flux d'information sont plutôt périodiques et l'aspect contrainte de temps (temps réel) est prioritaire.

- **Avantages des réseaux de terrain**

Le but initial des bus de terrain était de remplacer les anciens systèmes centralisés en distribuant le contrôle, le traitement des alarmes, le diagnostic aux différents équipements qui sont devenus de plus en plus intelligents.

Les anciens systèmes de communication industriels utilisaient la boucle de courant 4-20 mA, qui est un moyen de transmission analogique permettant d'envoyer un signal analogique sur une grande distance sans perte ou modification, pour relier les équipements aux machines de contrôle.[3]

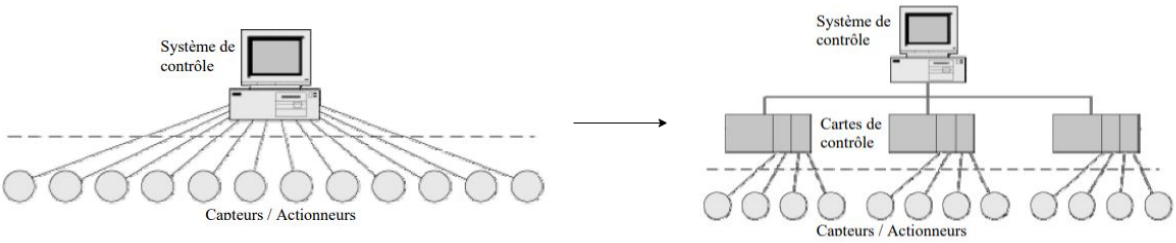

FIGURE 1.5 – Système de contrôle direct vers Systèmes de contrôle distribué

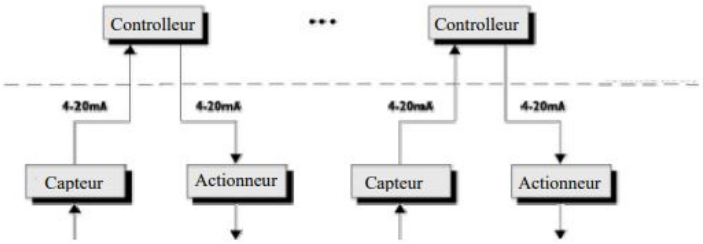

FIGURE 1.6 – Boucle de courant 4-20 mA

Après l'apparition de la communication numérique, cette technique a été rapidement remplacée par les bus de terrain. Leurs utilisation offre plusieurs avantages :

- **Réduction des coûts initiaux :**

- Réduction massive du câblage : un seul câble en général pour tous les équipements au lieu d'un cable par équipement.
- Possibilité de réutiliser le câblage analogique existant dans certains cas.
- Réduction du temps d'installation.
- Réduction du matériel nécessaire à l'installation.

- **Réduire le coût d'exploitation en :**

- Augmentant les performances de l'automatisme.
- Réduisant les coûts des extensions futures.

- **Réduction du coût de maintenance :**

- Complexité moindre donc moins de maintenance (fiabilité accrue).
- Maintenance plus facile : temps de dépannage réduit, localisation des pannes possibles grâce à des diagnostics en ligne donc à distance.
- Outils de tests dédiés.

- Flexibilité pour l'extension du bus de terrain et pour les nouveaux raccordements.

• **Performances :**

- Précision : la donnée numérique transférée est immunisée contre le bruit et la distorsion contrairement à un signal analogique.
- Les données et mesures sont généralement disponibles à tous les équipements de terrain.
- La Communication est possible entre deux équipements sans passer par le système de supervision.
- La structure distribuée permet de faire résider des algorithmes de contrôle au niveau de chaque équipement de terrain.[4]

1.3.1 Ethernet industriel

• **Protocole ethernet**

Ethernet est un protocole de réseau local à commutation de paquets. C'est une norme internationale : ISO/IEC 802-3.

Après avoir été utilisé pour les applications purement bureautiques, Ethernet est entré dans le monde industriel par le haut de la pyramide CIM. Dans un premier temps, il a servi à connecter les superviseurs aux automates ce qui a amélioré les performances tout en supprimant des cartes de communications spécifiques. Il a ensuite remplacé les anciennes connexions inter-automates.

Aujourd'hui il est descendu au niveau hiérarchique des bus de terrain et l'essentiel de la périphérie automate : IO devices, variateurs, codeurs, IHM … est disponible sous Ethernet. Pour résumer on peut affirmer que désormais la majorité des équipements compatibles avec les bus de terrain traditionnels tels que PROFIBUS, Modbus, DeviceNet, CANopen … existent également dans une version Ethernet industriel.

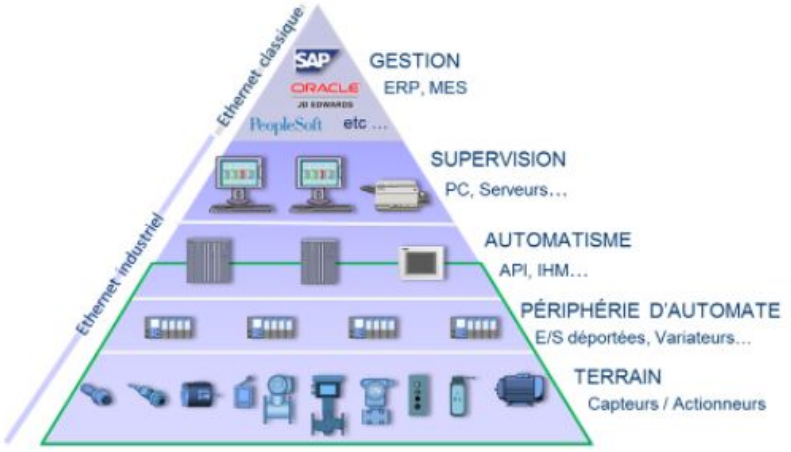

FIGURE 1.7 – Fabrication intégrée par ordinateur CIM

• **Comparaison entre Ethernet industriel et Bus de terrain classique**

l'Ethernet industriel dispose de sérieux atouts qu'il convient d'évaluer.

1- Vitesse : Le bus de terrain classique le plus rapide PROFIBUS fonctionne à la vitesse maximale de 12 Mb/s,alors que la plupart des protocoles basés sur l'Ethernet industriel dispose d'une vitesse de 100Mb/s. Un atout pour les applications exigeantes en temps de réponse.

2- Séparation galvanique : Sur un bus de terrain, tous les équipements d'un même segment utilisent le même média. Une anomalie sur un équipement, un connecteur ou le câble peut affecter la communication de l'ensemble des stations. Avec Ethernet, le média n'est pas partagé, une anomalie n'affectera que la station concernée par le lien.

3- Résistance aux perturbations électromagnétiques : Ethernet s'avère plus robuste qu'un bus de terrain par rapport aux perturbations électromagnétiques, ce qui ne veut pas dire qu'il est totalement insensible.

4- Cohabitation de protocoles : Il n'est pas possible de faire cohabiter sur un même câble des équipements PROFIBUS et Modbus même si les 2 fonctionnent sur RS485. Grâce à Ethernet, il est possible sur un même réseau de connecter par exemple des stations PROFINET et MODBUS-TCP tout en utilisant ce même réseau pour consulter une page web et envoyer un mail.

5- L'accès aux technologies IT : La plupart des équipements Ethernet industriel disposent d'un serveur Web. Un simple navigateur suffit pour configurer ou diagnostiquer l'appareil. Certains appareils peuvent envoyer automatiquement un mail pour signaler un défaut ou une opération de maintenance.

6- Accessibilité : La technologie Ethernet permet d'accéder beaucoup plus facilement à l'appareil y compris à distance, que s'il se trouvait sur un bus serial.[4]

1.3.2 Exemples de bus de terrain

Les différents bus de terrain offrent une multitude de caractéristiques et des performances variées. Il est difficile de faire une comparaison générale des performances des bus de terrain en raison des différences fondamentales dans la méthodologie de transfert des données. Dans le tableau comparatif ci-dessous, les différents bus de terrains avec leurs caractéristiques.

Bus de terrain	Bus power	Redondance	Max device	Synchronisation
EtherNet/IP	Non	Optionnelle	Illimité	Oui
Modbus	Non	Non	246	Non
PROFIBUS DP	Non	Optionnelle	126	Oui
PROFINET IO	Non	Optionnelle	Illimité	Non
EtherCAT	Oui	Oui	65,536	Oui
ControlNet	Oui	Oui	99	Non
AFDX	Oui	Oui	Illimité	Non
CANopen	Non	Non	127	Oui

TABLE 1.1 – Exemples bus de terrain

1.3.3 Profinet

PROFINET est l'évolution du réseau PROFIBUS DP vers une base Ethernet. On y retrouve certains des concepts de base qui ont permis à PROFIBUS de devenir le leader du marché (Simplicité de mise en œuvre, principe de diagnostic, fichiers descriptifs…) tout en améliorant les performances et en intégrant les services propres aux technologies IT.

- **Normes et standards**

Les spécifications PROFINET figurent dans les documents IEC 61158 type 10 et IEC 61784.

Par ailleurs PROFINET exploite des fonctions et services IT ayant déjà fait l'objet de standardisation pour des applications informatiques classiques (Web, FTP, RSTP, SNMP, 802.1Q ...).

• **Marché**

PROFINET est aujourd'hui une réelle alternative à PROFIBUS DP mais aussi aux autres bus de terrain traditionnels.

Du point de vue de la performance, la supériorité d'un réseau Ethernet tel que PROFINET est évidente. Cependant beaucoup d'installations n'ont pas un besoin immédiat d'amélioration en termes de vitesse, taille de données ou nombre de stations.

Si PROFINET séduit de plus en plus d'utilisateurs c'est que l'offre en matériel est de plus en plus large et que le prix d'une architecture Ethernet est désormais tout à fait compétitif.

Porté par ses atouts technologiques mais également par le support de grands acteurs de l'automatisme, PROFINET occupe la place de leader au sein des réseaux Ethernet industriels avec 29% de part de marché [5].

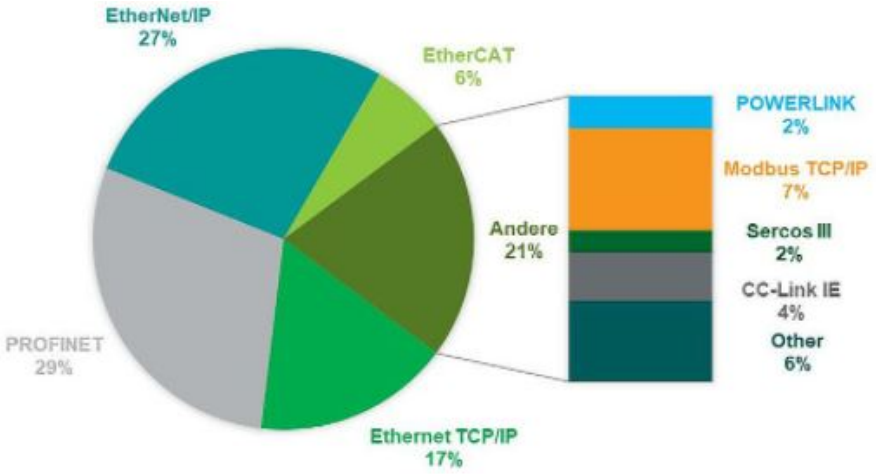

FIGURE 1.8 – Le marché mondial par protocole Ethernet 2018

Source :IHSMarkit|Technology

1.3.4 Communication

PROFINET échelonne la communication sur trois niveaux de performances :

• L'échange cyclique de données avec des contraintes de temps réel.

• L'échange de données acycliques pour la lecture et l'écriture de données selon les besoins (paramètres et données diagnostic).

• Un modèle d'alarme flexible pour la signalisation de défaillances avec trois niveaux d'alarme (besoin de maintenance, besoin de maintenance urgente et diagnostic).

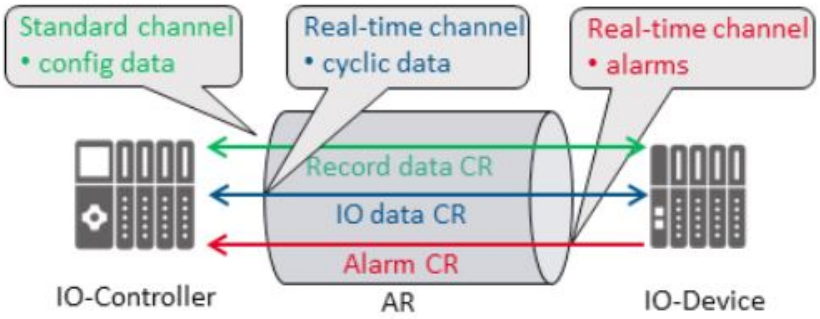

FIGURE 1.9 – Les CRs sur PROFINET selon le principe producteur/consommateur

1.4 Conclusion

Nous avons vu dans ce chapitre la constitution d'un système de contrôle-commande et abordé chaque composant faisant partie d'un SNCC ainsi que les bus de terrain. Étant donné la popularité et les performances du protocole PROFINET, notre choix s'est porté sur la conception d'un IO device qui sera apte à communiquer avec un PLC à l'aide de ce protocole.

Chapitre 2

Protocoles de communication PROFINET

2.1 Introduction

Profinet permet l'interfaçage direct d'appareils de terrain distribués sur l'Ethernet. Tous les appareils utilisés sont connectés sur le même réseau, et offrent donc une communication uniforme sur l'ensemble de l'usine de production. Profinet spécifie l'échange complet de données entre les IO controllers et les IO devices, ainsi que leurs paramétrages et leurs diagnostics. Il est conçu pour un échange de données rapide avec des cycles de quelques millisecondes, et est basé sur un modèle fournisseur/consommateur.

2.2 Équipements et intégration :

2.2.1 Constituants d'un réseau PROFINET

PROFINET suit le modèle fournisseur/consommateur pour l'échange de données. Cela signifie que l'IO controller et L'IO device envoient des données cycliques indépendamment.
Les classes d'appareils se présentent comme suit :

- **IO controller**

Il s'agit généralement d'un automate programmable (PLC) dans lequel s'exécute le programme d'automatisation. Le IO controller fournit des données de sortie aux IO devices configurés, en tant que fournisseur et est le consommateur d'entrées venant de celui-ci

- **IO device**

Un IO Device est un appareil de terrain distribué qui échange des données avec un ou plusieurs IO Controllers en utilisant les mécanismes de Profinet. Une configuration Profinet contient au moins un IO Device.

- **IO supervisor**

Une station d'ingénierie permettant la configuration et le diagnostic d'un réseau PROFINET, typiquement un logiciel sur PC [6]

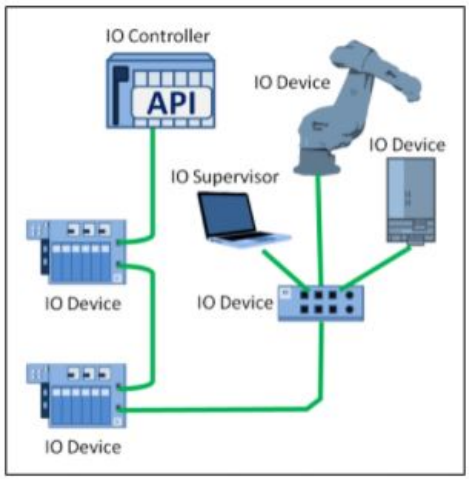

FIGURE 2.1 – Les composants d'un réseau PROFINET

2.2.2 Intégration des équipements PROFINET

À chaque équipement PROFINET (IO Device) est obligatoirement associé, un fichier descriptif de type GSD qui comporte les informations nécessaires pour que l'IO Supervisor puisse les intégrer dans I'IO Controller. Les différentes étapes de l'ingénierie se présentent comme suit :

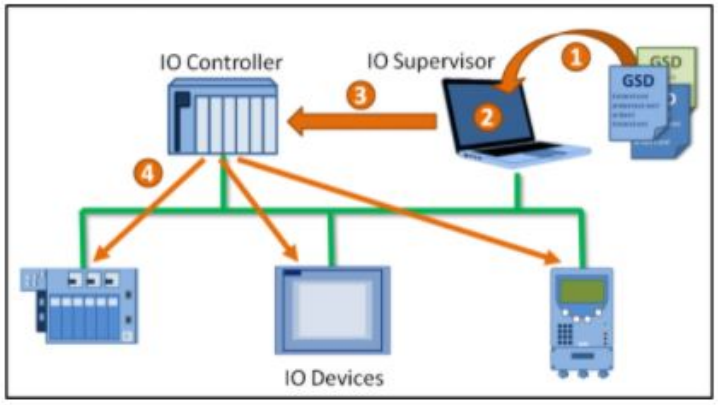

FIGURE 2.2 – Intégration d'un IO Device PROFINET

1- Les fichiers descriptifs (GSD) des différents équipements sont chargés dans la bibliothèque de l'IO Supervisor.

2- La configuration du réseau PROFINET et paramétrage des IO Devices.

3- Le transfert du projet dans l'IO Controller.

4- L'IO Controller vérifie la présence et l'identité des IO Devices et leur envoie leurs paramètres. Les stations passent ensuite en échange de données. [5]

2.2.3 Modules et submodules

Les modules sont positionnés dans des slots, les submodules dans des subslots. Le remplacement, sans reconfiguration au préalable, des modules et des sous-modules est donc possible. La différence entre un dispositif "modulaire" et un dispositif "compact" est simplement que les fentes avec leurs modules/sous-modules sont fixes dans un dispositif compact. Les adresses de device/slot/subslot peuvent être appliqués à n'importe quelle architecture de dispositifs [7] .

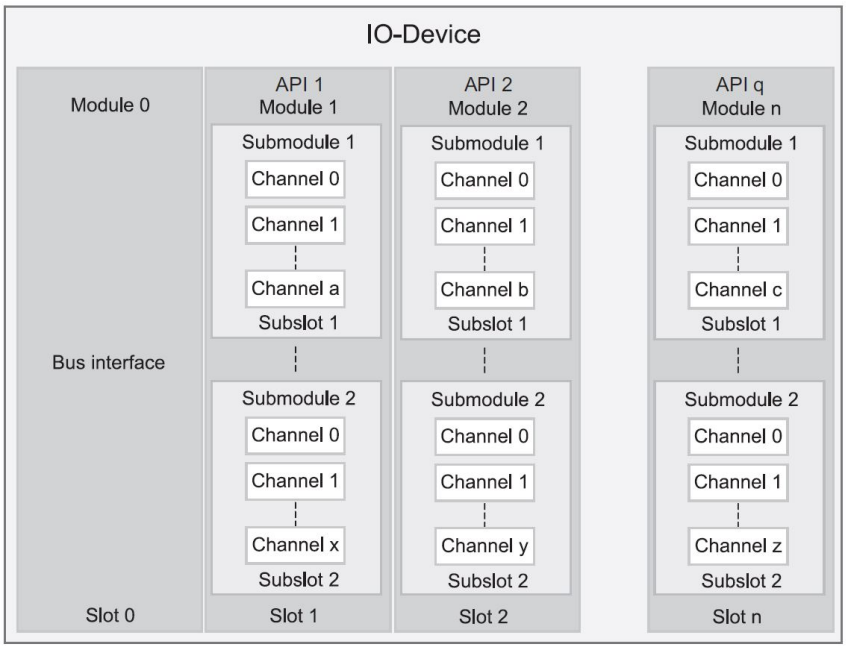

FIGURE 2.3 – Modèle IO device

2.2.4 GSD file

Chaque fournisseur d'un end device doit fournir un fichier GSD associé. Ce fichier décrit les propriétés de l'appareil concerné. cependant, le fichier est présent dans un fichier XML. Dans Profinet, le terme Generic Station Description Markup Language (GSDML) est également utilisé.

La structure et les règles d'un fichier GSD sont définies par le schéma GSDML qui est administré par Profibus International et qui contient les règles de validité qui permettent, par exemple, de vérifier la syntaxe d'un fichier GSD.

```
+--ProfileHeader
+--ProfileBody
   |
   +--DeviceIdentity
   |  |
   |  +--InfoText
   |  +--VendorName
   |
   +--DeviceFunction
   |  |
   |  +--Family
   |
   +--ApplicationProcess
      |
      +--DeviceAccessPointList
      +--ModuleList
      +--SubmoduleList
      +--ValueList
      +--LogBookEntryList
      +--CategoryList
      +--ChannelDiagList
      +--ChannelProcessAlarmList
      +--UnitDiagTypeList
      +--ExternalTextList
```

2.2.5 Conformance classes

PROFINET offre un grand nombre de fonctions. Ces fonctions sont divisées en classes de conformité de manière claire. Elles fournissent un résumé pratique des différentes fonctionnalités de l'IO device.

Il existe trois Conformance Classes (CC) :

•La Conformance Class A (CC-A) fournit des fonctions de base pour PROFINET avec une communication en temps réel (RT). Tous les services informatiques peuvent être utilisés sans restriction. Les applications typiques se trouvent, par exemple, dans l'automatisation des entreprises. La communication sans fil est spécifiée pour cette classe.

• La classe de conformité B (CC-B) étend PROFINET pour inclure des diagnostics de réseau utilisant des mécanismes IT et des informations de topologie du réseau.

• Conformance Class C (CC-C) décrit les fonctions de base pour les appareils avec

réservation de bande passante et synchronisation supportées par le matériel. La communication Isochronous Real-Time (IRT) est disponible.

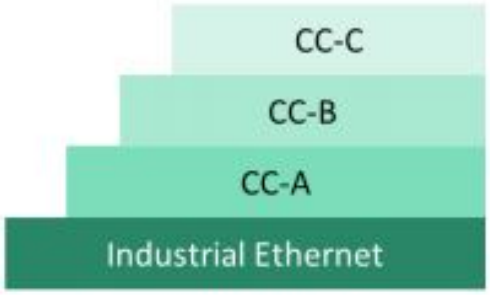

FIGURE 2.4 – Structure de Conformance Classes

2.3 Communication PROFINET :

Les échanges entre IO device et IO controller utilisent différents canaux selon le type de données :

- Canal temps réel pour les E/S cycliques et les alarmes .
- Canal standard UDP/IP pour le paramétrage, la configuration .

2.3.1 Modèle OSI et profinet

Le protocole PROFINET respecte l'organisation en 7 couches du modèle OSI.

- Pour la communication en temps réel (cyclique et alarmes), on utilise directement la couche ethernet sans utiliser les autres couches , et cela afin de minimiser le temps d'encapsulation du paquet et de respecter les contraintes de temps réel.
- Le canal standard utilisé ,par exemple pour configurer une station ou faire un diagnostic (acyclique), est basé sur la couche Réseau IP et la couche Transport TCP/UDP en plus d'ethernet, la contrainte de temps réel est négligeable.
- L'équipement peut aussi avoir un trafic IT classique sans rapport avec PROFINET, par ex. accéder à la page web de l'équipement.

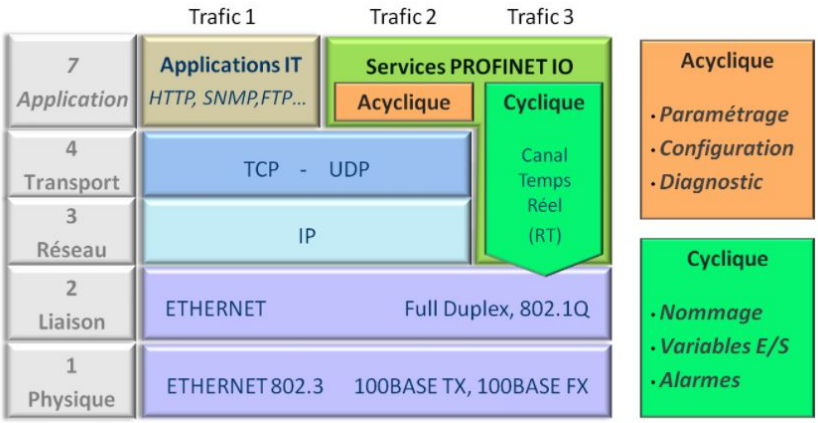

FIGURE 2.5 – Modèle OSI dans PROFINET

2.3.2 Temps réel

Le temps réel signifie qu'un système traite les événements externes dans un temps défini. Si la réaction d'un système est prévisible, on parle d'un système déterministe. Les exigences générales pour la communication en temps réel (RT) sont donc :

- Une réponse déterministe.

- Des temps de réponse définis, généralement jusqu'à 5ms pour les applications standard.

Étant donné que la tâche principale du processeur est l'exécution du programme utilisateur et non l'échange de données, la communication en temps réel sur Ethernet ne doit entraîner qu'une charge insignifiante pour le processeur. Dans le même temps, le cycle d'envoi, c'est-à-dire la durée d'une transmission de données de l'application d'un dispositif A à l'application d'un appareil B, doit être aussi faible que possible. La figure 2.6 montre les facteurs responsables du cycle d'envoi.

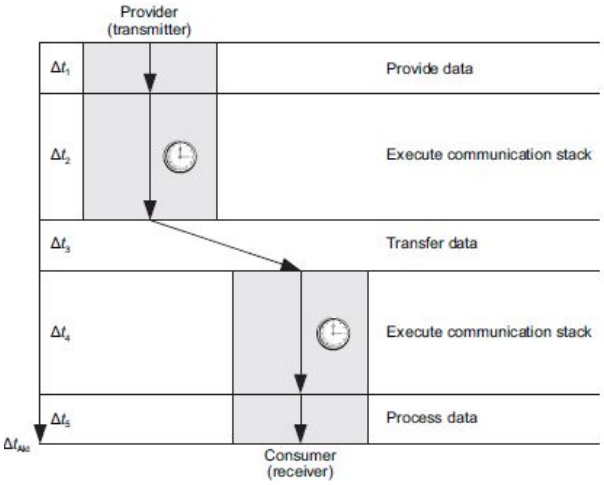

FIGURE 2.6 – Composition du cycle d'envoi

L'une des possibilités de mise en œuvre de la communication en temps réel est l'utilisation de protocoles de communication standard tels que TCP/IP. Cependant, leur application est également associée à des inconvénients : l'encapsulation des trames augmente leurs longueurs, et entraîne donc une augmentation du temps de transmission sur la ligne (Fig. 2.7).

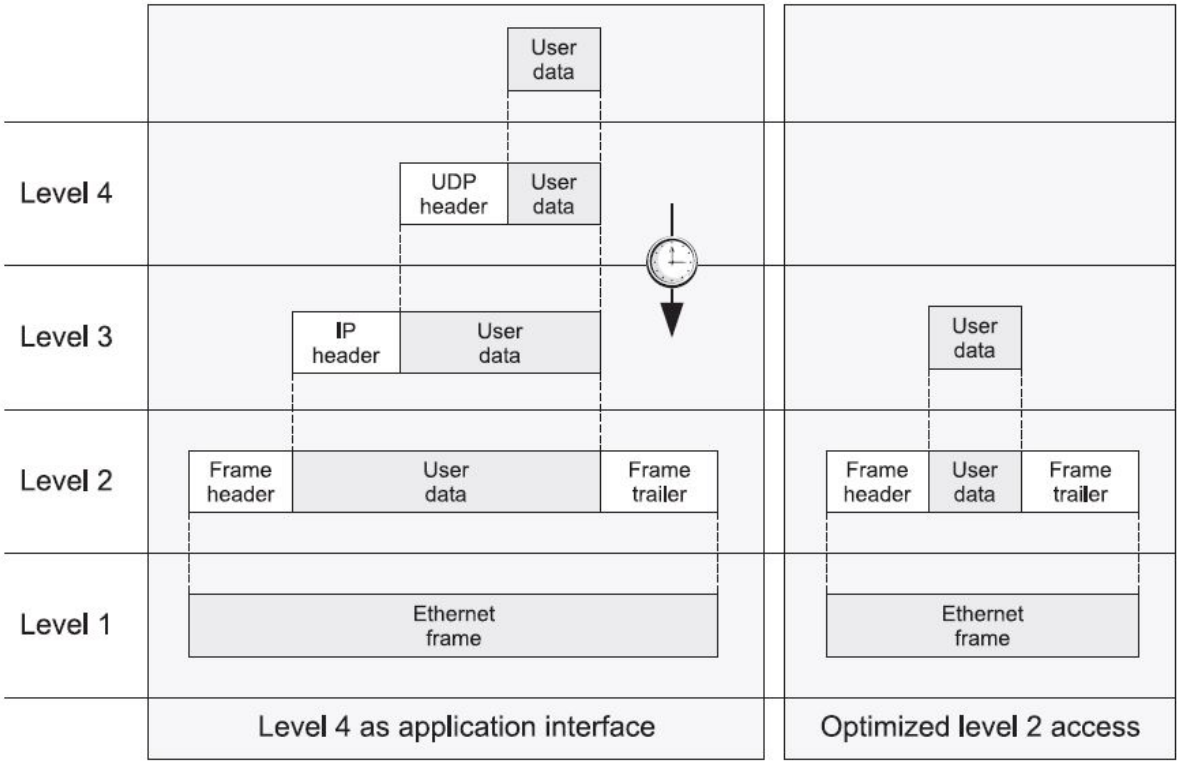

FIGURE 2.7 – Origine de l'overhead lors de l'encapsulation des données

Le protocole temps réel permet la transmission performante de données cycliques et de messages contrôlés par des événements (alarmes). Il est divisé en trois classes (RTC : real-time classes) :

- Classe temps réel 1 (RTC 1) : convient à la transmission de données cycliques et acycliques. Aucune exigence particulière n'est imposée aux switchs utilisés.

- Classe temps réel 2 (RTC 2) : convient pour la transmission d'interruptions et de données cycliques. Des switchs spéciaux doivent être utilisés dans ce cas.

- Classe temps réel 3 (RTC 3) : convient à la transmission de données cycliques avec des applications de contrôle du mouvement. Des switchs spéciaux doivent également être utilisés avec RTC 3, et une planification explicite de la communication doit être effectuée à l'avance.

2.3.3 Ethernet

Le standard Ethernet est le protocole de la 2eme couche (liaison de donnée) le plus répandu dans les réseaux locaux (LAN).

• Trame Ethernet de la 2eme couche :

La trame Ethernet est une unité de données du protocole de la couche liaison de données et utilise les mécanismes de transport de la couche physique Ethernet sous-jacente. , sa trame se présente sous cette forme :

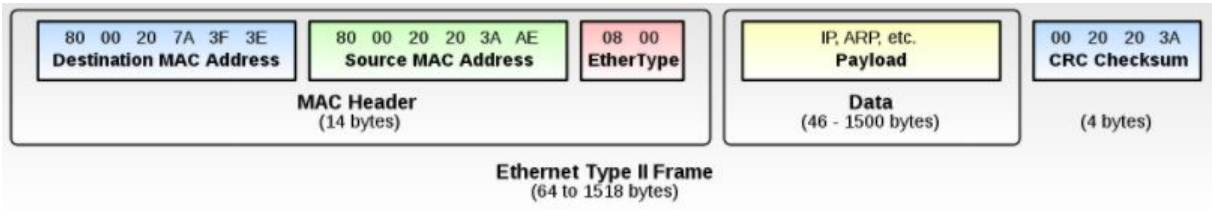

FIGURE 2.8 – Trame Ethernet

- Chaque interface Ethernet possède une adresse fixe et unique intégrée par le fournisseur. Cette adresse, appelée adresse matérielle ou MAC (Media Access Control), est enregistrée sur la carte réseau et sert à l'identifier dans le réseau local.

- La case Ethertype nous permet de connaitre le type de trame envoyée ou reçue, voici quelques exemples :

< 0x0600 : IEEE802.3 length block

0x0600 : Ethernet II type block

0x0800 : IP

0x0806 : ARP

0x8100 : Tag Control Information ; data packets with VLAN-TPID

0x8892 : Profinet

0x88E3 : MRP

0x88CC : LLDP

- La partie Data contient les données envoyées.

- CRC (Contrôle de redondance cyclique) permet de détecter des erreurs de transmission.

2.3.4 Éléments du protocole Profinet

La communication en temps réel est basée sur le protocole ethernet de la deuxième couche, la structure de la trame se présente sous cette forme :

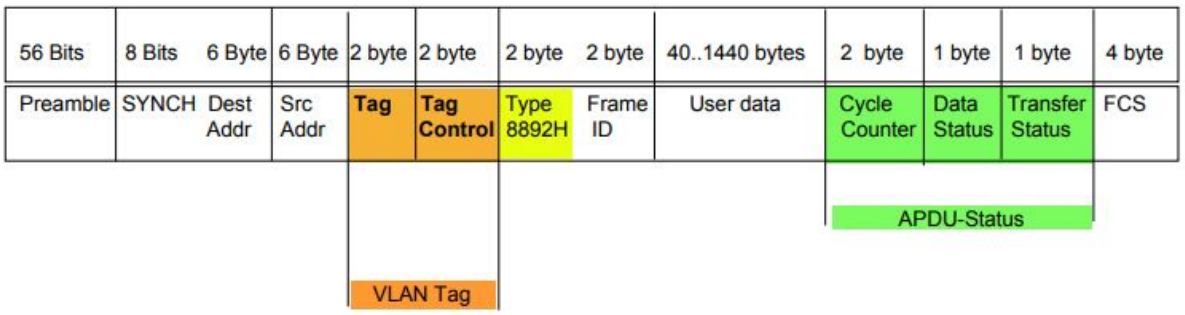

FIGURE 2.9 – Construction d'une trame PROFINET temps réel

Afin de maintenir un faible retard de la trame pendant la communication, la transmission de tous les services cycliques et acycliques se voient attribuer des priorités par l'application du format de trame VLAN (balisage VLAN). À l'aide de l'élément de protocole "priorité de l'utilisateur". La "priorité de l'utilisateur" peut avoir des valeurs comprises entre 0 (priorité la plus basse) et 7 (priorité la plus élevée). Les trames Profinet en temps réel sont envoyées avec la priorité 6 ou 7.

élément du protocole	Exemples et explications
Ethernet II header	Adresse MAC de la destination et de la source
VLAN-TAG	**VLAN Tag Protocol Identifier** : **User Priority** : 0x00 : IP, DCP 0x01 - 0x04 : Reservé 0x05 : RTA low-priority or UDP RTA low-priority 0x06 : RTC 1/2, UDP RTC (RT over UDP), 0x07 : PTCP, MRP, MRRT **CFI (Canonical Format Indicator)** : 0 : Ethernet 1 : Token Ring **VLAN ID** : 0x00 : Transmission de donnée avec priorité 0x001 : Standard setting 0x002-0xFFE : For free use 0xFFF : Reservé
Ethertype	0x8892 : Profinet 0x8100 : Tag Control Information VLAN-TPID
frameID	RTC1, UDP RTC : 0xC000 – 0xF7FF : Unicast frames 0xF800 – 0xFBFF : Multicast frames DCP : 0xFEFD : GetReq, SetReq, GetResp, SetResp 0xFEFE : IdentifyReq 0xFEFF : IdentifyResp
payload	**User data** : dépend de l'IO device **APDU Status (Application Protocol Data Unit Status)** : **Cycle counter** :Le compteur est incrémenté par le provider à chaque envoi. **Data status** **Transfer status**

TABLE 2.1 – Éléments du protocole profinet

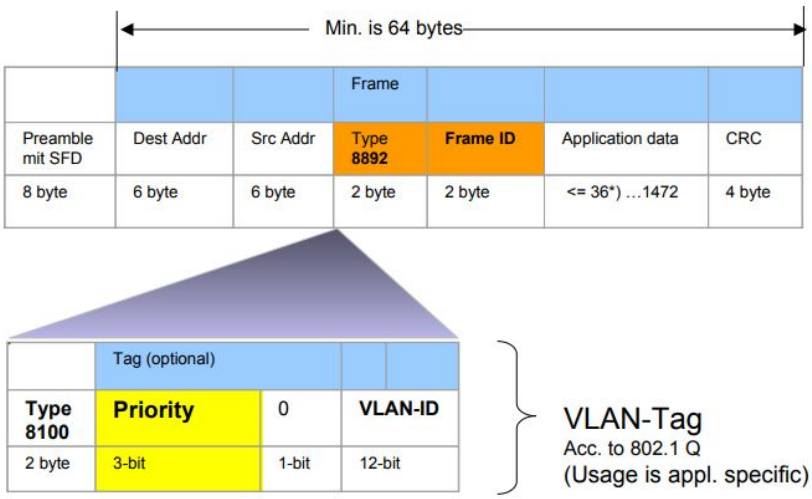

FIGURE 2.10 – Priorisation de la trame avec le tag VLAN

2.3.5 TCP/UDP

TCP et UDP s'exécutent au-dessus de l'IP et se fondent sur les services fournis par ce dernier.

•TCP (Transport Control Protocol) assure un service de transmission de données fiable avec une détection et une correction d'erreurs de bout en bout.

•UDP (User Datagram Protocol) offre un service de transmission de datagrammes sans connexion.

Avec TCP ou UDP, il est possible de remettre des données à des processus d'application s'exécutant sur une machine distante. Ces processus d'application sont identifiés par numéros de port.

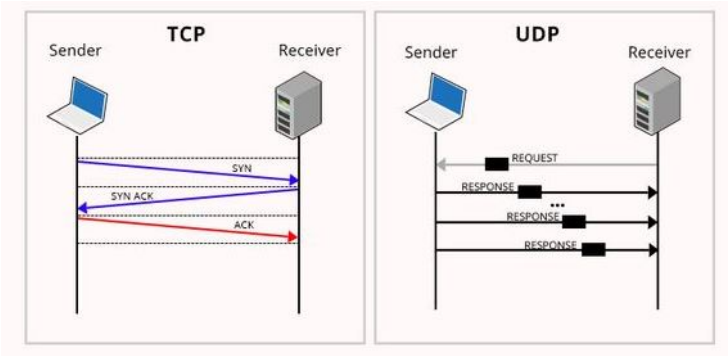

FIGURE 2.11 – TCP vs UDP communication

le protocole PROFINET communique dans les échanges de données acycliques à l'aide d'un canal UDP.

2.4 Principaux canaux

2.4.1 Application relation (AR)

Une "Application Relation" (AR) est un élément logique et virtuel qui permet l'échange de données entre deux dispositifs sur des canaux de communication. La transmission effective des données a lieu à l'intérieur de ces canaux au moyen d'une ou de plusieurs "communications relations" (CR).

Plusieurs relations d'application peuvent être établies vers un IO Device à partir de différents IO controllers.

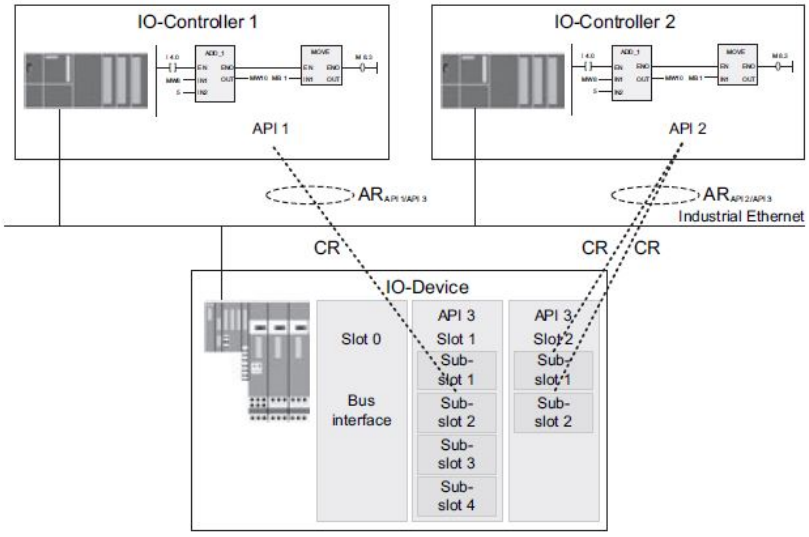

FIGURE 2.12 – Application relations (AR) et connexion entre IO controller et IO device

2.4.2 Communication relation (CR)

Plusieurs relations de communication (CR) peuvent être établies au sein d'un AR. Les types de CR suivants existent :

- Le CR de données d'E/S pour la transmission cyclique de données .

- Le CR Record data pour la transmission acyclique de record pour la mise en service, le paramétrage, les diagnostics, etc.

- L'alarme CR pour la transmission acyclique d'événements.

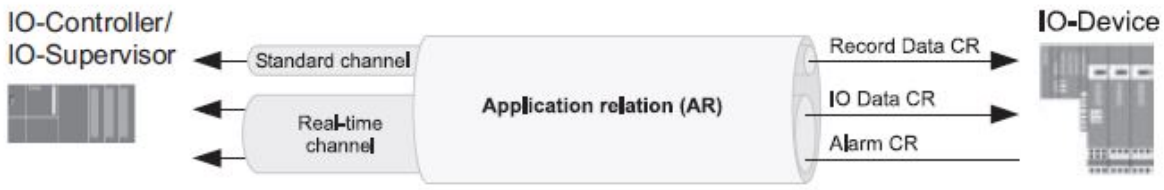

FIGURE 2.13 – Échange de donnée dans un CR

2.4.3 Communication cyclique

Les données IO cycliques sont transmises via le "IO data CR" en tant que données en temps réel entre le fournisseur et le consommateur dans un laps de temps paramétrable. Les temps de rafraichissement peuvent être spécifiés individuellement pour les connexions aux différents appareils et sont ainsi adaptés aux exigences de l'application. De même, différents temps de rafraichissement peuvent être sélectionnés pour les données d'entrée et de sortie dans une plage de 250 µs à 512 ms.

Après le démarrage du système, les données d'E/S sont échangées cycliquement entre le IO controller et l'IO device.

Chaque élément de données d'E/S contient deux attributs, les IOPS (IO Provider Status) et IOCS (IO Consumer Status), qui permettent aux IO controller et IO device d'évaluer la fiabilité des données.

L'IOPS est transmis par le fournisseur en même temps que les données et est donc dépendant de son état.

En revanche, l'IOCS peut seulement être renvoyé par le consommateur au fournisseur qu'après la réception des données.

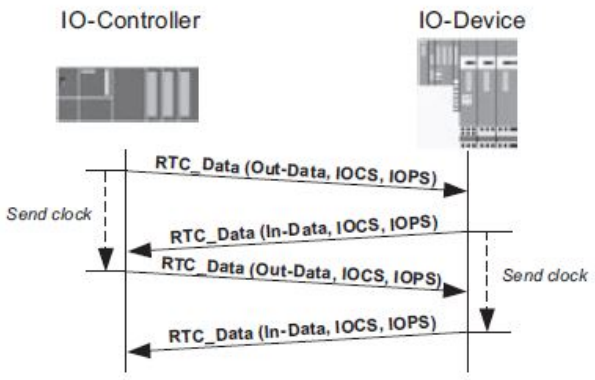

FIGURE 2.14 – Transmission de données cycliques avec le protocole RT pour Profinet

IOCS et IOPS peuvent avoir les valeurs GOOD et BAD représentés dans la trame en hexadécimal par 0x80 ou 0x00.
Comme norme, les sorties de l'IO controller sont toujours nommées output, et ceux de l'IO device input, vice versa.

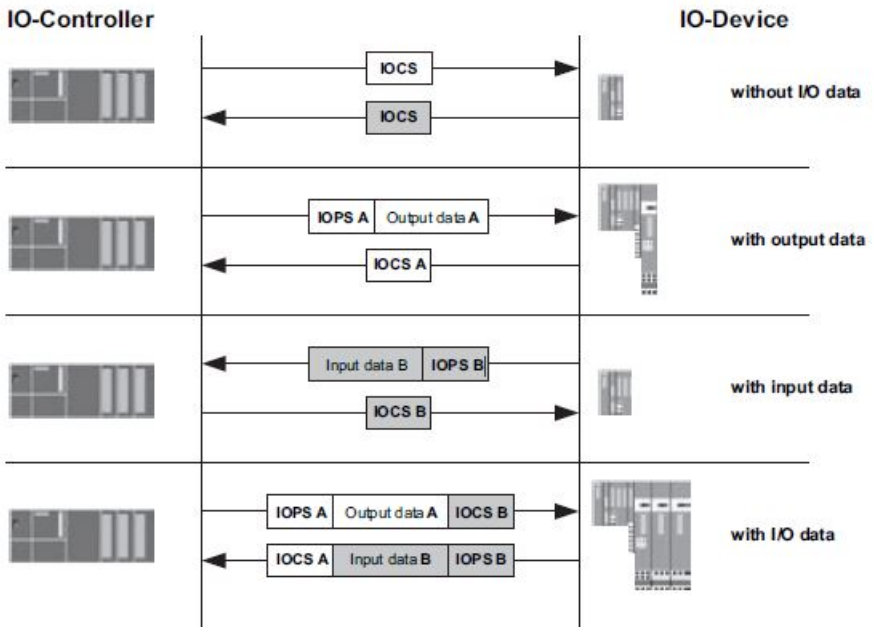

FIGURE 2.15 – Transmission de données cycliques avec le protocole RT pour Profinet

2.4.4 Communication acyclique

La transmission de données acycliques sert à échanger des données qui ne sont pas critiques en termes de temps. Les services de lecture et d'écriture (Read/Write record) sont utilisés. L'échange de données acycliques a lieu dans le canal de communication NRT (NRT : non-real-time) en utilisant le protocole remote procedure calls (RPC), basé sur UDP. Les services de lecture/écriture se composent toujours d'une demande et d'une réponse ultérieure.

L'échange acyclique de données à l'aide de la fonction "record data CR" peut être utilisé pour :

•La configuration des paramètres et d'autres configurations des IO devices.

•La lecture des informations d'état (Statut) et des données d'identification et de maintenance (I&M) permettant une identification unique des appareils, modules et sous-modules et de leurs versions.

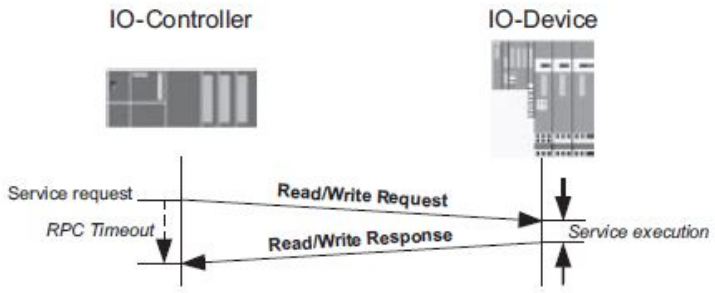

FIGURE 2.16 – Services acyclique read/write services Profinet IO

IM0	Main function	HW/FW version
IM1	TAG_FUNCTION TAG_LOCATION	System designation, location designation
IM2	INSTALLATION_ DATE	Installation date
IM3	DESCRIPTOR	Comment
IM4	SIGNATURE	Signature
IM5	Communication modul	HW/FW version

FIGURE 2.17 – Liste des I&Ms

2.4.5 Alarmes

Les alarmes sont des données acycliques en temps réel qui doivent être envoyées dans un délai défini.

Ce concept d'alarme couvre à la fois les événements définis par le système (tels que le retrait et l'insertion de modules) et la signalisation de défauts détectés par l'IO controller (par exemple, une tension, charge défectueuse ou une rupture de fil).

L'utilisateur peut définir des alarmes de processus correspondantes pour les messages du processus, par exemple un seuil de pression dépassé. Dans ce cas, le IO device peut encore être opérationnel. Ces alarmes de processus peuvent avoir une priorité différente de celle des alarmes de diagnostic.

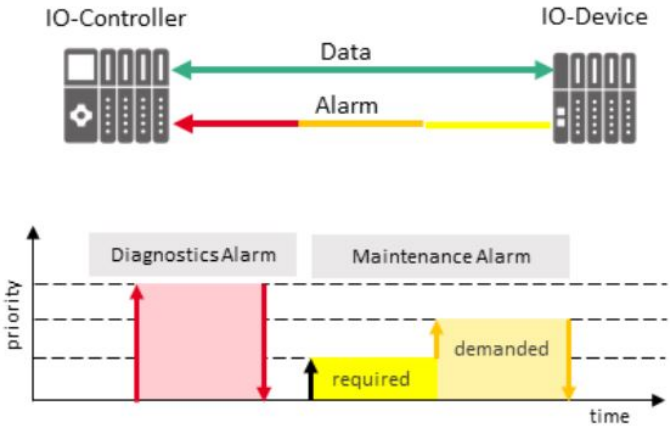

FIGURE 2.18 – Alarmes avec différentes priorités

Exemple de type d'alarmes :

•Process alarm : L'alarme signale l'apparition d'un événement provenant d'un processus, par exemple le dépassement d'un seuil de pression.

•Pull : l'alarme signale le retrait d'un module/sous-module.

•Plug : l'alarme signale l'insertion d'un module/sous-module.

Émetteur	Protocole	Contenu
IO-device	PN-AL	Alarm notification high, 1 byte user data
IO-controller	PN-AL	ACK-RTA-PDU
IO-controller	PN-AL	Alarm ack high, alarm type Process, slot, subslot, Status OK
IO-device	PN-AL	ACK-RTA-PDU

TABLE 2.2 – Étapes d'envoi d'une alarme

2.5 Démarrage (startup)

2.5.1 Context management (CM)

Le rôle du context management (CM) est de gérer les AR et CR. Il s'occupe de :
- L'initialisation des applications relations (AR).
- L'initialisation des communications relations (CR).
- La définition de paramètres de communication adéquats pour les relations de communication, par exemple les délais et les modes de fonctionnement.
- Paramétrage de l'Ethernet device driver (EDD).
- Distribution des paramètres décrits dans le fichier GSD.

2.5.2 Le lancement en différentes étapes

Dans un réseau PROFINET, chaque appareil reçoit un nom symbolique qui identifie de manière unique l'appareil. L'appareil est identifié et configuré avec ce nom au cours du processus d'ingénieurie. Les adresses MAC et IP exactes sont configurées à l'aide de ce nom lorsque l'application PROFINET est lancée.

•Attribution d'un nom à un IO device :

L'IO device se voient attribuer un nom avant l'établissement effectif d'une connexion. Le nom est attribué par un IO supervisor et enregistré en mode permanent dans l'IO device. Le nom est utilisé pour l'identification univoque d'un IO device pendant le runtime.

•Au démarrage d'un appareil Profinet IO (avant que l'adresse IP ne soit définie), le protocole DCP est utilisé.

1- L'automate envoie un message de diffusion (broadcast) DCP.

2- Tous les IO devices du sous-réseau répondent en indiquant leur adresse MAC.

3- L'automate envoie un message DCP à l'IO device avec une adresse MAC spécifique, contenant l'adresse IP et le nom de la station que l'IO device doit utiliser.

4- L'IO device définit son adresse IP et son nom de station en conséquence et confirme en envoyant une requête DCP à l'IO controller.

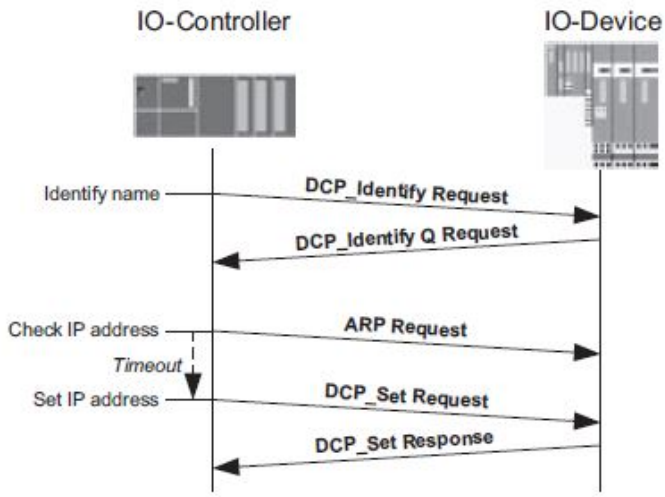

FIGURE 2.19 – Attribution adresse IP à l'aide du protocole DCP

•Établissement de la connexion

Un CR de données IO est établi par une séquence de connexion entre l'IO controller et l'IO device. La demande d'écriture suivante lance son paramétrage, qui se termine par la demande DControl suivante. Suite à la réponse positive à la demande CControl envoyée par l'IO device, la connexion est établie.

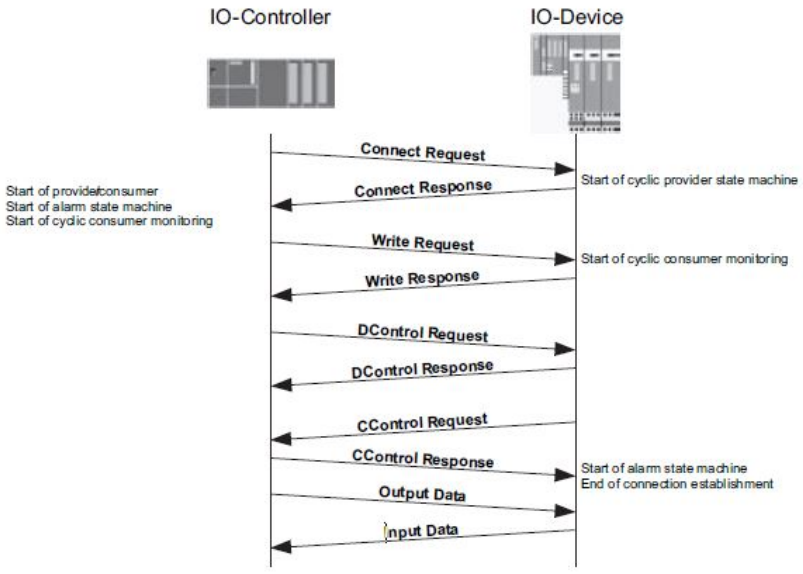

FIGURE 2.20 – Etablissement de la connexion CM (Context management)

•Trames Ethernet envoyées pendant le démarrage :

Dans cet exemple, l'IO controller est démarré en premier, puis l'IO device.

Sender	Protocol	Content
IO-controller	LLDP	Nom, MAC, IP addr, port (envoyé chaque 5 secondes)
IO-controller	PN-DCP	“Ident req”. cherche “rt-labs-dev” (envoyé chaque 2.5 seconds)
IO-controller	ARP	Qui utilise mon IP ?
IO-device	LLDP	Nom, MAC, IP addr, port (envoyé chaque 5 secondes)
IO-device	PN-DCP	Station name,IP addr, gateway,vendeur r(chaque 3 seconds)
IO-device	PN-DCP	“Ident OK” Identify response.
IO-controller	PN-DCP	“Set Req” Set IP request. Utilise IP addr et gateway.
IO-device	PN-DCP	“Set OK” Status.
IO-controller	ARP	Qui a <IO-device IP address> ?
IO-device	ARP	IP <IO-device IP address> est à <IO-device MAC address>
IO-controller	PNIO-CM	“Connect request” Controller MAC
IO-device	PNIO-CM	“Connect response” MAC address, UDP port
IO-device	PNIO-PS	FrameID 0x8001. Cycle counter, provider stopped. 40 bytes data.
IO-controller	PNIO-PS	FrameID 0x8000. Cycle counter, provider running. 40 bytes data.
IO-controller	PNIO-CM	“Write request” API, slot, subslot, data.
IO-device	PNIO-CM	“Write response” API, slot, subslot, status.
IO-controller	PNIO-CM	“Control request” (DControl). Command : ParameterEnd.
IO-device	PNIO-CM	“Control response” Command : Done
IO-device	PNIO-PS	FrameID 0x8001. Cycle counter, provider running. 40 bytes data.
IO-device	PNIO-CM	“Control request” (CControl). Command : ApplicationReady
IO-controller	PNIO-CM	“Control response” Command : ApplicationReadyDone

TABLE 2.3 – Trames envoyées pendant le démarrage

2.5.3 Protocoles utilisés

•LLDP (Link Layer Discovery Protocol) :

Pendant le démarrage, les appareils commencent à échanger des informations LLDP et les informations de topologie peuvent être lues. Cette topologie sera enregistré dans l’appareil de terrain. Si des changements dans la topologie sont détectés, le contrôleur recevra une alarme de la station affectée.

•DCP (Discovery and basic Configuration Protocol) :

Il est utilisé par l'IO controller pour découvrir et identifier les informations sur les IO devices et configurer les paramètres tels que le nom du périphérique PROFINET et l'adresse IP sur un réseau PROFINET. PROFINET DCP est un protocole de couche de liaison Ethernet et offre de multiples services tels que 'Identify All', 'Identify', 'Set', Set – 'Flash', Set – 'Reset to Factory', 'Get' et 'Hello'.

-DCP Identify All : Il nous permet d'identifier/naviguer dans le réseau PROFINET et de trouver tous les appareils PROFINET attachés.

-DCP Identify : Il est utilisé lorsqu'un appareil doit être trouvé à l'aide d'un nom d'appareil particulier ou connu.

-DCP Set : Il est utilisé pour définir le nom ou l'adresse IP de l'appareil.

-DCP 'Set / Reset to Factory' : Le service 'Set / Reset to Factory' est une commande spéciale qui place l'appareil dans un état d'usine PROFINET (par défaut) qui est un nom vide ("") et des paramètres IP de 0.0.0.0.

-DCP Get : Le service "Get" est utilisé pour obtenir des informations d'un appareil.

Ce tableau nous permettra de distinguer le type de service de la trame DCP :

Type de service	Description
0	Request
1	Response Success

TABLE 2.4 – Type de service DCP

Service ID	Description
3	Request
4	Response Success
5	Response Success

TABLE 2.5 – Service ID DCP

• ARP (Address Resolution Protocol) :

ARP est un protocole Internet qui relie une adresse IP à une adresse physique (adresse MAC Ethernet). Chaque station disposant d'une adresses IP dans le réseau gère une table de résolution d'adresse à cette fin. Les paires déjà attribuées d'adresses IP et MAC sont implicitement stockées dans cette table, appelée cache ARP.

2.6 Conclusion

Dans ce chapitre, nous avons présenté le fonctionnement du protocole PROFINET que ce soit en communication, les étapes du démarrage, ainsi que différentes notions qui constitueront une base pour la suite du projet. Pour la conception de notre IO device, nous nous limiterons à un device de classe B communiquant en RT1 (real time class 1).

Chapitre 3

Implémentation du protocole PROFINET

3.1 Introduction

Pour exploiter les fonctionnalitées du protocoles PROFINET de façon optimale, Nous allons implémenter sur un système embarqué un IO device exécutant des tâches sur les canaux cycliques , acycliques et des alarmes, cette application aura la spécificité de transmettre les données lues d'un capteur en temps réel sur le canal cyclique, paramétrer le temps entre chaque lecture du capteur et un seuil sur le canal acyclique, qui en cas de dépassement , engendrera une alarme qui sera lancée en temps réel sur le canal adéquat.

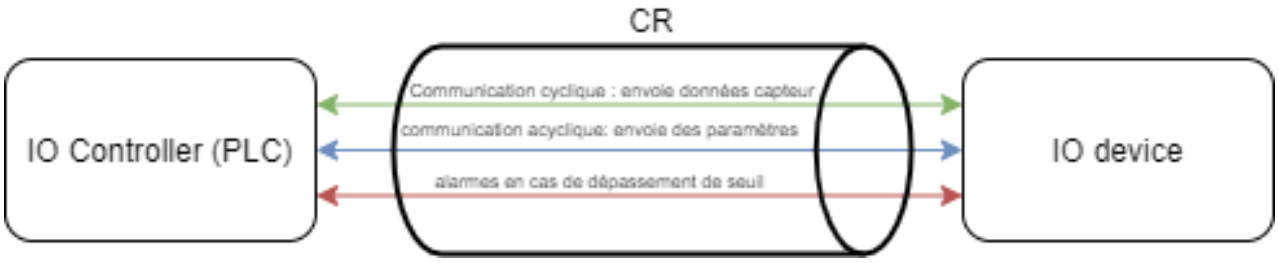

FIGURE 3.1 – Canaux PROFINET et utilisation dans l'application

L'implémentation se fera sur embedded linux en utilisant la librairie p-net, il est cependant possible de l'utiliser sur un microprocesseur bare-metal ou à l'aide d'un RTOS mais cela nécessite l'utilisation d'un RTOS fourni par rt-labs, ou l'adaptation des API de celui-ci avec un RTOS standard (FreeRTOS par exemple).

L'une des problématiques rencontrées est l'adaptation du système d'exploitation linux pour qu'il respecte les contraintes de temps réel.

3.2 Hardware utilisé

L'implémentation d'un IO device nécessite un hardware qui dispose des capacités minimales suivantes :

• Il doit être possible d'envoyer et de recevoir des trames Ethernet brutes.

• Il doit être possible de stocker des données entre deux exécutions, dans un système de fichiers ou une mémoire non volatile.

• Pour la CC (Class conformance) B, il doit y avoir une implémentation SNMP que le stack p-net peut utiliser.

Par rapport au port ethernet, voici la configuration matérielle requise :

•Au moins 100 Mbit/s full duplex

•Les câbles standards et croisés doivent être supportés.

•Auto-polarité.

•Auto-négociation. [8]

Le matériel doit être capable d'ordonnancer les multitâches en temps réel, (dans notre cas , une distribution linux) afin d'obtenir des performances optimales.

3.2.1 Choix du microcontrôleur

Après une étude de faisabilité, notre choix s'est porté sur une Raspberry pi 3 model B, de par son accessibilité et ses caractéristiques, c'est la carte idéale pour prototyper une application.

L'intégration d'un système d'exploitation nous facilite le développement et le déploiement de notre application.

FIGURE 3.2 – Raspberry pi 3 model B

3.2.2 Caractéristiques

La puissance de calcul ainsi que les caractéristiques de notre hardware ont une importance par rapport à la mise en place d'un système respectant les contraintes de temps réel.

-Processeur

• Broadcom BCM2387 chipset.

• 1.2GHz Quad-Core ARM Cortex-A53 (64Bit)

-Mémoire

• 1GB LPDDR2

Système d'exploitation

• Boots à partir d'une Micro SD card, distribution linux raspbian avec un noyau temps réel.

-Alimentation

• Micro USB socket 5V1, 2.5A

-Ethernet

• Pour la partie ethernet, la raspberry utilise la microchip LAN9514, qui contient un hub USB 2.0 haut débit avec quatre PHY USB 2.0 en aval entièrement intégrés, un PHY USB 2.0 intégré en amont , un contrôleur MAC/PHY 10/100 Ethernet et un contrôleur EEPROM, ce qui est suffisant pour notre application.

[9]

3.3 Linux sur embarqué

3.3.1 Introduction

La stabilité et la fiabilité de Linux sur embarqué a été démontrée au fil des années, plusieurs produits ont été développés en utilisant ce système d'exploitation, ses nombreux avantages ont pallié auw contraintes de mémoire. Open source et documenté, il peut être modifiable à notre guise pour notre application, dans notre cas, nous allons l'adapter pour qu'il réponde à nos contraintes de temps réel.

3.3.2 Caractéristiques

Voici quelques-unes des caractéristiques importantes de Linux, et des systèmes d'exploitation de style Unix en général.

3.3.2.1 Multitâche

- L'ordonnanceur Linux implémente un véritable multitâche préemptif, dans le sens où un processus de priorité plus élevée, rendu prêt par l'occurrence d'un événement asynchrone va préempter le processus en cours d'exécution. Cependant, le noyau Linux lui-même n'est pas préemptible. Ainsi, un processus ne peut pas être préempté pendant qu'il exécute un service du noyau.

3.3.2.2 Multi-utilisateur

-Il existe un certain nombre de fonctionnalités qui prennent en charge la confidentialité des données. Linux préserve cette caractéristique et la met à profit dans les environnements de serveurs et des systèmes embarqués.

3.3.2.3 Multithreading

- Linux offre un support étendu pour le véritable multitraitement symétrique (SMP). Un thread ressemble fortement à un processus fils classique à la différence qu'il partage beaucoup plus de données avec le processus qui l'a créé :
-Les variables globales.
-Les variables statiques locales.
-Les descripteurs de fichiers (file descriptors).
Le multi-threading est donc une technique de programmation permettant de profiter des avantages (et aussi de certaines contraintes) de l'utilisation des threads.

3.3.2.4 Système de fichiers hiérarchique

- Tous les systèmes d'exploitation modernes ont des systèmes de fichiers hiérarchiques. Mais le modèle Linux/Unix ajoute des fonctionnalités en plus de ce à quoi nous sommes habitués avec les systèmes d'exploitation traditionnels.

3.3.2.5 Device drivers

- Un pilote est une portion de code exécutée dans l'espace du noyau. Il est chargé de faire l'interface entre un programme utilisateur et un composant matériel.

3.3.2.6 La contrainte de temps

Il y a deux types de contraintes de temps pour les systèmes embarqués : douce ou dure.
- Une contrainte douce (système temps réel doux) est moins contraignante car en cas d'erreur raisonnable, le système reste fonctionnel. processus aurait dû s'exécuter.
- Par opposition, la contrainte dure ne permet aucune erreur liée au moment d'exécution, cela empêchera le bon fonctionnement du système. [10]

3.3.2.7 La capacité du réseau

Il définit si un système peut être relié à un réseau. Linux offre la possibilité de se connecter à un réseau à l'aide des nombreux drivers disponibles tel que "net" qui contient le code source des différents protocoles réseau supportés par le noyau Linux. Nous pouvons citer ipv4, ipv6 ou x25 [11] .

3.3.3 Propriétés temps réel de Linux :

Linux n'est pas un système d'exploitation en temps réel. le protocole profinet a des contraintes de temps assez strictes. Par exemple, un automate Simatic (avec les paramètres par défaut) enverra une alarme si aucune trame Profinet n'est reçue pendant 7-8 ms. Il est donc important que Linux soit suffisamment réactif.

Il y a quelques méthodes qui peuvent être utilisées pour améliorer la réactivité de Linux. Nous citerons toutes les méthodes utilisées afin de pallier à la contrainte de temps réel.

3.3.3.1 Option "SCHED_FIFO"

Le noyau linux offre deux méthodes d'ordonnancement en temps réel, l'ordonnancement FIFO (First in First out) SCHED_FIFO et le Round Robin SCHED_RR . La

méthode que nous avons utilisé est la méthode SCHED_FIFO qui implémente l'algorithme first in first out, quand une tâche SCHED_FIFO s'exécute, elle continue son exécution jusqu'à ce qu'elle se termine, se fasse bloquer par le processeur, l'appel système yield ou sa préemption par une tache en temps réel d'une priorité supérieur. Contrairement au round robin, il n'y a pas d'intervalle de temps pour chaque tâche (timeslice).

Pour notre application : Nous avons selectionné l'option d'ordonnancement FIFO du noyau Linux. Ceci est fait en passant l'argument de ligne de commande -DUSE_SCHED_FIFO=ON à cmake. [8]

3.3.3.2 Application sur un cœur de processeur séparé

Il est possible d'indiquer au noyau Linux de ne pas placer de processus sur un cœur de processeur spécifique. On doit avoir plus d'un cœur dans notre CPU.
Dans le fichier :

```
sudo nano /boot/cmdline.txt
```

On ajoute :

```
isolcpus=2
```

Puis on reboot le système

En définissant la commande de démarrage Linux isolcpus=2, le noyau ne placera pas de processus automatiquement sur le noyau numéro 2 du processeur. Cette option est généralement définie à partir du bootloader. La meilleure façon de vérifier quels sont les cœurs de CPU qui sont actuellement isolés :

```
cat /sys/devices/system/cpu/isolated
```

•On place l'application Profinet sur le noyau CPU isolé. Cela se fait en utilisant :

```
sudo taskset -c 2 ./pn_dev
```

où -c 2 indique le noyau CPU à utiliser.

3.3.3.3 Patches en temps réel

En appliquant le patch en temps réel (PREEMPT_RT), les propriétés en temps réel peuvent être améliorées. Pour que les patchs en temps réel aient un effet sur p-net,

l'option de cmake USE_SCHED_FIFO doit être active.

Afin d'accomplir cela, nous avons compilé le noyau linux avec l'extension PREEMPT-RT. Ensuite, nous avons installé l'image du noyau ainsi que les modules nécessaires.

```
~$ cd /tmp
/tmp$ tar xzf rt-kernel.tgz
/tmp$ cd boot
/tmp/boot$ sudo cp -rd * /boot/
/tmp/boot$ cd ../lib
/tmp/lib$ sudo cp -dr * /lib/
/tmp/lib$ cd ../overlays
/tmp/overlays$ sudo cp -d * /boot/overlays
/tmp/overlays$ cd ..
/tmp$ sudo cp -d bcm* /boot/
```

On ajoute ceci au fichier /boot/config.txt :

```
kernel=vmlinuz4.14.74-rt44-v7+
```

Après le redémarrage du système, on peut confirmer la bonne installation du kernel :

```
uname -r
4.14.74-rt44-v7+
```

la figure 3.3 représente la latence résultante du linux kernel Standard et Preempt-RT Kernels (3b) [12] .

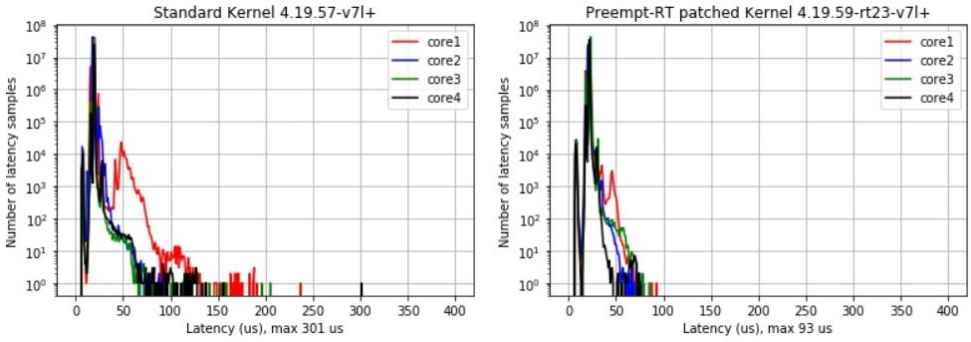

FIGURE 3.3 – Comparaison de latence entre un noyau standard et un noyau temps réel

3.3.3.4 Augmenter le temps de cycle de l'application

Pour les tests, on peut augmenter le temps de cycle de l'automate afin de réduire les problèmes de time-out. Le nombre autorisé de trames manquées peut également

être augmenté dans les paramètres du PLC.

3.3.3.5 Matériel de l'interface réseau

Si notre interface réseau Ethernet est connectée via USB, il peut y avoir une latence supplémentaire. Cela peut affecter l'intervalle des trames Profinet transmises, dans notre cas le matériel qui concerne l'interface réseau ne cause pas de problème.

3.4 Bibliothèque p-net

La bibliothèque Profinet p-net de RT-Labs est utilisée pour l'implémentation d'IO device Profinet. elle est facile à utiliser et occupe un faible espace de mémoire. Elle est particulièrement bien adaptée aux systèmes embarqués où les ressources sont limitées et où l'efficacité est cruciale. La bibliothèque est fournie avec les fichiers sources complets, y compris les couches de portage.

3.4.1 Fonctionnalités

Écrit en C, le stack p-net offre aux utilisateurs toutes les tâches basiques du protocole PROFINET telles que l'implémentation du temps réel, les alarmes , l'échange de données cycliques et acycliques, le lancement de l'application avec les protocoles LLDP et ARP, ainsi que l'envoi de packet TCP/IP.
p-net nous offre beaucoup de possibilités d'un point de vue fonctionnel, parmi eux :
- La mise en oeuvre d'un device de CC (Conformance Class) A ou B
- L'envoi des trames de RTC 1 (Real Time Class 1)
- La disponibilité sur Linux, RTOS ou bare metal.
- L'utilisation de plusieurs ports ethernet (sur linux)
- L'activation du protocole SNMP
- Un nombre configurable de modules et submodules virtuels
- La disponibilité des données I&M (I&M0 - I&M4).

3.4.2 Limitations

Malgré toutes les fonctionnalités listées précedemment, le stack p-net est dédié aux IO devices, il ne peut donc pas être utilisé pour les IO controllers.
Son utilisation nécessite la connaissance des différentes limites du stack qui sont :
- L'absence de redondance de média (ne supporte pas le protocol MRP).
- Le stack ne supporte pas la RT_CLASS_UDP, la DHCP, l'utilisation de plusieurs IO controller en même temps.
- L'option fast startup qui nous permet le lancement rapide de l'application n'est pas disponible.
- L'option IRT (Isochronous real time) n'est pas implémentée donc il n'y a pas de synchronisation du temps.
- Les alarmes n'utilisent pas de paquets UDP (Il n'y a que le mécanisme de base implémenté c'est à dire l'envoi d'alarme en temps réel).

3.4.3 Dépendances

- **cmake 3.14**

CMake fournit un ensemble d'outils permettant de compiler un projet pour différentes plateformes, de faire des tests et de créer des packages pour différents systèmes.

Le principe d'une compilation à l'aide de CMake est d'écrire des fichiers CMakeLists.txt dans tous les répertoires qui contiennent du code ou documentation à compiler/installer. La coutume est alors de créer un sous-répertoire build, dans lequel on va compiler notre package puis d'y appeler cmake. On dit alors qu'on construit le projet "out-of-source", tout les fichiers temporaires seront dans le sous-répertoire ROOT/build qu'on pourra supprimer si besoin sans affecter notre projet. cmake va alors créer des fichiers de compilation dépendants de l'architecture/OS.
- On compile généralement en dehors du répertoire source et on le laisse donc intact.
- On utilise une seule syntaxe relativement simple.
- L'outil est cross-plateform, il tourne aussi bien sous Linux que sous MacOS et Windows.

Pour utiliser CMake, on doit :

- Définir des fichiers CMakeLists.txt à placer dans le répertoire source.
- Faire appel à cmake pour qu'il génère les makefiles.
- Faire appel à make pour compiler le projet.

[13]

FIGURE 3.4 – CMake logo

- **gcc 4.6**

GNU Compiler Collection, abrégé en GCC, est un ensemble de compilateurs créés par le projet GNU. GCC est un logiciel libre capable de compiler divers langages de programmation, dont C, C++, Objective-C, Java, Ada, Fortran et Go.

FIGURE 3.5 – GCC GNU

- **net-snmp**

Pour la CC B (conformance class B) le protocole SNMP est nécessaire. Linux utilise net-snmp (BSD License)

3.5 Implémentation p-net

Nous allons implémenter le stack profinet ainsi que notre application sur linux embarqué, et pour cela, nous allons suivre les procédures citées ci-dessous

3.5.1 Compilation

En premier lieu, nous devons installer les dépendances nécessaire à l'importation et la compilation de notre projet.

- Afin de compiler le stack nous devons installer la dernière version de cmake :

```
sudo apt update
sudo apt install snapd
sudo reboot
sudo snap install cmake --classic
```

- On vérifie si c'est bien la dernière version qui est installée :

```
cmake --version
```

Il est important que la version soit supérieure à 3.14
Pour pouvoir importer la librairie et faire la gestion de version pour le travail en collaboration avec l'entreprise, nous devons installer git :

```
sudo apt install git
```

Afin d'importer la librairie à partir de git, nous devons la télecharger ou l'importer directement de git :

```
git clone --recurse-submodules https://github.com/rtlabs-com/p-net.git
```

Puis on crée et configure le build à l'aide des options cmake :

```
cmake -B build -S p-net -DUSE_SCHED_FIFO=ON
```

Après la création du fichier "build", on lance le build du code à l'aide de cette commande :

```
cmake --build build --target install
```

On accède au fichier build :

```
cd build
```

On ne doit pas oublier l'activation de l'interface ethernet et son initialisation :

```
sudo ifconfig eth0 192.168.0.50 netmask 255.255.255.0 up
```

On lance l'application comme suit :

```
sudo taskset -c 2 ./pn_dev -v
```

3.5.2 Configuration

Le fichier CMakeLists.txt contient un ensemble de directives et d'instructions décrivant les fichiers sources et les cibles du projet (exécutable, bibliothèque, ou les deux).

Pour configurer les différentes options de l'application

```
set(PNET\_MAX\_AR                    2
  CACHE STRING "Number of connections. Must be > 0. 'Automated RT Tester'
    uses 2")

```

Le code sera automatiquement complété dans option.h comme suit :

```
#if !defined (PNET\_MAX\_AR)\newline
/** Number of connections. Must be > 0. "Automated RT Tester" uses 2 */
#define PNET\_MAX\_AR                @PNET\_MAX\_AR@
#endif
```

3.6 Application spécifique

3.6.1 Introduction

L'application consistera en l'implémentation d'un IO device qui capte les valeurs du capteur associé (un capteur de température dans notre cas), et envoie sur le canal cyclique les valeurs de la température lues.
Sur le canal acyclique, il y'aura envoi d'un paramètre considéré par l'application comme un seuil, qui en cas de dépassement, enverra une alarme au PLC.
Nous allons en premier lieu écrire le GSD file associé à notre IO device puis expliquer les différentes parties du code, ce qui nous permettra de modifier les parties nécessaires pour notre application en utilisant les fonctions de l'API PROFINET.

3.6.2 GSD file pour l'IO device

Le GSD file décrit l'IO device et permet à l'ingénieur qui met en place le SCNN de connaitre les données envoyées ainsi que les paramètres à partir de l'IO supervisor, Il n'aura qu'à ajouter le GSD file à son environnement de développement.

Afin d'adapter le GSD file de l'IO device à notre application, nous devons ajouter ceci :

```
<DataItem DataType="Float32" TextId="p_value" UseAsBits="false"  />

<Ref DataType="Float32" ByteOffset="0" DefaultValue="1" AllowedValues
   ="0..99" Changeable="true"
<Ref DataType="Float32" ByteOffset="0" DefaultValue="2" AllowedValues
   ="0..999" Changeable="true"
```

La partie IOData décrit les données envoyées dans le canal cyclique en temps réel, le format "Float32" correspond à la norme IEEE 754 sur l'arithmétique à virgule flottante. Elle est la norme la plus employée actuellement pour le calcul des nombres à virgule flottante avec les CPU et les FPU.

FIGURE 3.6 – Norme IEEE 754

Les paramètres sont représentés dans la <RecordDataList>, Nous avons 2 paramètres, l'un représentant le temps entre la lecture du capteur, et l'autre le seuil à ne pas dépasser avant l'envoi d'une alarme, ils sont de type Float32.

3.6.3 Structure du code

3.6.3.1 Organisation des fichiers

Le code se présente sous cette forme :

```
/p-net
├── CMakeLists.txt
├── include
├── sample_app
└── src
    ├── common
    ├── device
    └── ports
```

Comprendre la structure du code nous permettra de l'adapter dans le futur et de comprendre ce que nous devons changer pour l'implémentation de notre application. Le code est divisé en 3 parties :

-Port :

Cette partie s'occupe de l'interfaçage entre le hardware et la partie application, elle contient une base pour le networking, c'est à dire la création de socket, l'envoi de trame sur la couche liaison de données et de paquets UDP. Elle s'occupe aussi de tout ce qui est manipulation de fichier (sauvegarde, chargement, comparaison ...) , ainsi que la base du système d'exploitation. Les fichiers et les fonctions de cette partie auront un préfixe pnal_ et feront partie de la couche d'abstraction la plus basse.

Exemple d'envoi de trame ethernet sur la deuxième couche :

```
1 handle->socket = socket (PF_PACKET, SOCK_RAW, htons (linux_receive_type));
2 bind (handle->socket, (struct sockaddr *)&sll, sizeof (sll));
3 int ret = send (handle->socket, buf->payload, buf->len, 0
```

[14]

-Device :

Cette partie implémente les fondements de la bibliothèque, les fichiers et fonctions de cette partie ont le préfixe pf_ , elle est implémentée sous forme de machine d'état où est intégré le CM (context management) , ainsi que des fonctions qui permettent d'ajouter à un buffer les headers et données nécessaires , qui seront utilisées plus tard dans la constitution de la trame cyclique et de l'alarme, en plus des paquets acycliques.

-API :

Cette partie utilise les fonctions du device pour réaliser les fonctionnalités finales

de notre application, elle nous propose des fonctions qu'on utilisera pour réaliser les tâches nécessaires au bon fonctionnement de notre IO device.

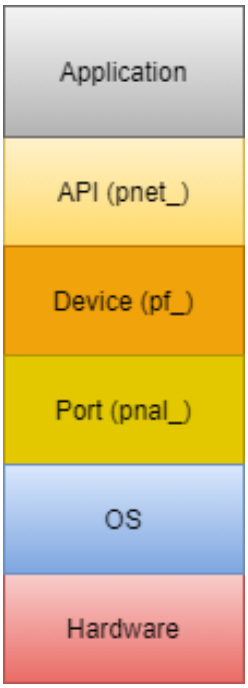

FIGURE 3.7 – Couche du programme

3.6.3.2 Machine d'état

Il existe plusieurs façons de programmer une application, on peut directement utiliser nos fonctions dans une boucle et tout s'exécutera séquentiellement, c'est ce qu'on appelle une superloop, les inconvénients de cette manière de programmmer sont évidents : le manque de contrôle sur les timings et l'absence d'ordonnancement. On a donc inventé les machines d'états, qui nous permettent de gérer les conditions de transition vers un autre état et ainsi avoir plus de maitrise sur les échéances de notre système, il y'a aussi l'utilisation d'un OS (système d'exploitation) qui nous donne l'option de préemption et d'ordonnancement, ce qui nous permet le multithreading, c'est à dire l'execution de plusieurs threads simultanément. Dans un système complexe, toutes les architectures possibles peuvent coexister tant que le bon fonctionnement de l'application est assuré.

Dans le cadre de notre application, la librairie utilise plusieurs machines d'états qui s'occupent du lancement de l'application, de la mise en place des différentes configurations et du context management, et de la communication cyclique, acyclique

et des alarmes.L'application est donc lancée sous forme de thread avec un timer ce qui nous permet de profiter des avantages d'un système d'exploitation.

Les machines d'états sont présentées comme suit :

•Alarmes :

- ALPMI Alarm Protocol Machine Initiator. initialise l'alarme pour être envoyée au IO controller

- ALPMR Alarm Protocol Machine Responder. Réponse de l'IO controller à l'alarme (from IO-controller)

- APMR Acyclic Protocol Machine Receiver. Reçoit les trames Ethernet d'alarme de haute et de basse priorité en provenance de l'IO device

- APMS Acyclic Protocol Machine Sender. Envoie des trames Ethernet d'alarme au IO controller

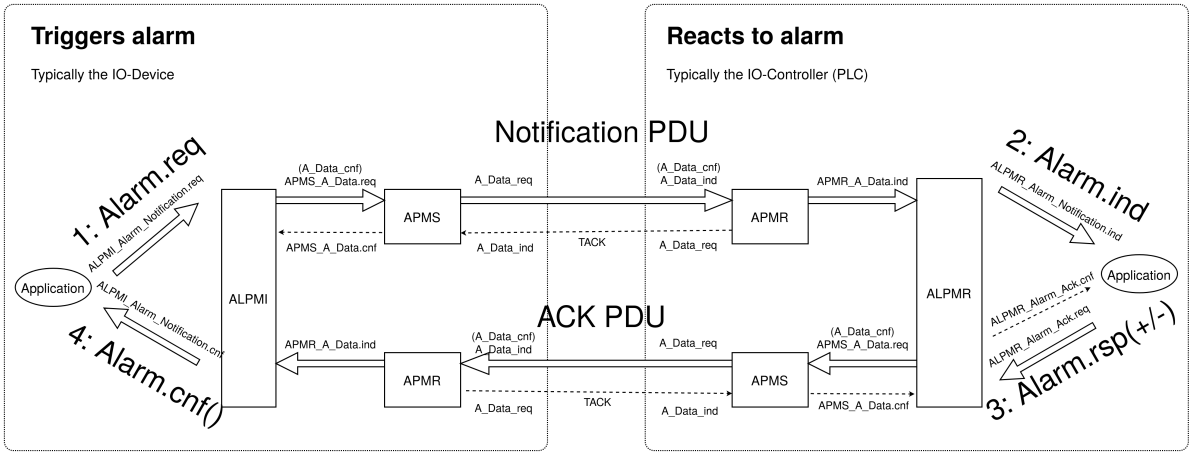

FIGURE 3.8 – Machine d'état alarme

•CMDEV Context Management protocol machine Device, gère l'établissement de la connexion pour les IO devices :

Il existe 11 états dans ce CM :

- POWER_ON, Initialisation des données.

- W_CIND, Attend l'indication de connexion (in the connect UDP message)

- W_CRES, Attend la réponse de connexion de l'application et le démarrage du CMSU.

- W_SUCNF, Attend la confirmation CMSU .

- W_PEIND, Attend l'indication PrmEnd (dans le message UDP DControl).

- W_PERES, Attend la réponse PrmEnd de l'application.
- W_ARDY, Attend que l'application soit prête à partir de l'application.
- W_ARDYCNF, Attend la confirmation du contrôleur que l'application est prête.
- WDATA, Attend qu'un échange de données cycliques soit établi.
- DATA, Échange de données et monitoring des connexions.
- ABORT, Interrompt la relation d'application.

•CMDMC Multicast s'occupe de la communication entre IO devices à l'aide de messages DCP.

•CMINA Context Management Ip and Name Assignment protocol machine :

Cette machine d'état est responsable de l'attribution du nom de la station et de l'adresse IP. Elle effectue une réinitialisation d'usine lorsque le IO controller le demande.

Il existe 4 états dans ce CM :

- SETUP
- SET_NAME
- SET_IP
- W_CONNECT

Elle aide au traitement des demandes DCP Set et DCP Get.

•CMIO Context Management Input Output protocol machine.

•CMPBE Context Management Parameter Begin End protocol machine.

•CMRDR Context Management Read Record Responder protocol machine, répond aux paramètres lus par l'IO controller.

•CMRPC Context Management RPC Protocol Machine, gère la communication RPC via UDP.Elle s'occupe des services suivants :

- Connect
- Release
- DControl ("Parameter end" est envoyé au IO-Device)
- CControl ("Application ready" est envoyé au IO-Controller)
- Paramètre read (Utilise CMRDR)
- Paramètre write

•CMSM Context Management Surveillance protocol Machine, surveille l'établissement d'une connexion.

Le composant CMSM surveille l'établissement d'une connexion. Une fois que le dispositif entre dans l'état DATA, la mission de ce composant est terminée.

Il s'agit essentiellement d'une minuterie, qui a deux états : IDLE et RUN. Si elle n'est pas arrêtée avant son expiration, le stack entre dans l'état PNET_EVENT_ABORT. Le minuteur revient à l'état IDLE au bout du compte. En général, le délai d'attente est d'environ 20 secondes (il peut être ajusté par le IO controller).
Le minuteur est démarré à PNET_EVENT_STARTUP (au message de demande de connexion), et arrêté à PNET_EVENT_DATA.
Il surveille également les messages de réponse et d'indication :
-Read
-Write
-DControl
Il démarre le timer à l'envoi du message "response" et l'arrête à la réception du message "indication".
•CMSU Context Management Start Up machine. Démarre d'autres machines d'état, par exemple PPM et CPM.
•CMWRR Context Management Write Record Responder protocol machine
•CPM Consumer Protocol Machine, pour la réception des données cycliques :
Reçoit les données cycliques. Vérifie que les données entrantes respectent le protocole et que la synchronisation des trames entrantes est correcte. Stocke les données entrantes dans un buffer.
Plusieurs instances de CPM peuvent être utilisées en parallèle.
Il existe 3 états dans ce CM :
-W_START Attente d'initialisation
-FRUN
-RUN
S'il y a un délai d'attente dans l'état RUN, il repasse à l'état W_START.
•FSPM Fieldbus Application Layer Service Protocol Machine. Interaction avec l'application de l'utilisateur ; implémente des callbacks.
•PPM Provider Protocol Machine, pour l'envoi de donnée cyclique.
Il existe 2 états dans ce CM :
-W_START
-RUN
[8]

3.6.4 API PROFINET

Voici un bref aperçu de certaines des fonctions les plus importantes de l'API du stack Profinet de p-net :

Cette fonction s'occupe de l'initialisation :

```
pnet_t *pnet_init(const pnet_cfg_t *p_cfg)
```

Cette fonction sert à initialiser les ARs

```
int pnet_application_ready (pnet_t *net, uint32_t arep)
```

L'application signale qu'elle est prête à échanger des données avec l'envoi d'une demande de CControl au IO controller.

Cette fonction initialise la communication cyclique :

```
void pnet_handle_periodic (pnet_t *net)
```

Cette fonction permet à l'application de demander l'interruption de la connexion.

```
int pnet_ar_abort (pnet_t *net, uint32_t arep)
```

Cette fonction permet l'envoi des données cycliques et les IOPS au IO controller :

```
int pnet_input_set_data_and_iops (pnet_t *net, uint32_t api, uint16_t slot,
    uint16_t subslot, const uint8_t *p_data, uint16_t data_len, uint8_t
   iops)
```

Cette fonction est utilisée pour récupérer l'état du consommateur IOCS d'un sub-slot.

```
int pnet_input_get_iocs (pnet_t *net, uint32_t api, uint16_t slot, uint16_t
    subslot, uint8_t *p_iocs)
```

Cette fonction sert à lire les données cycliques envoyées par l'IO controller vers l'IO device ainsi que l'état du fournisseur IOPS.

```
int pnet_output_get_data_and_iops (pnet_t *net, uint32_t api, uint16_t slot
   , uint16_t subslot, bool *p_new_flag, uint8_t *p_data, uint16_t *
   p_data_len, uint8_t *p_iops)
```

Cette fonction définit l'état de l'IO device quand il est consommateur :

```
int pnet_output_set_iocs (pnet_t *net, uint32_t api, uint16_t slot,
   uint16_t subslot, uint8_t iocs)
```

Alarmes :

Cette fonction permet d'émettre une alarme à partir de l'IO device vers l'IO controller

```
int pnet_alarm_send_process_alarm (pnet_t *net, uint32_t arep, uint32_t api
   , uint16_t slot, uint16_t subslot, uint16_t payload_usi, uint16_t
   payload_len, const uint8_t* p_payload)
```

Cette fonction envoie un ACK (acknowledge) au contrôleur. Elle doit être appelée par l'application après avoir reçu une alarme dans le rappel pnet_alarm_ind().

```
int pnet_alarm_send_process_alarm (pnet_t *net, uint32_t arep, uint32_t api
    , uint16_t slot, uint16_t subslot, uint16_t payload_usi, uint16_t
    payload_len, const uint8_t* p_payload)
```

3.6.5 Implémentation de la communication cyclique

3.6.5.1 Lecture capteur

Afin de lire les données du capteur, nous allons écrire dans un fichier qui sera lu par l'application avant chaque envoi cyclique [15] . Voici la fonction qui nous permettra de lire les valeurs du fichier à partir de l'application, elle est implémentée dans sample_app.c :

```
uint8_t *read_value_sensor (const char* file_name) {

FILE *fp = fopen(file_name, "r");

   if(fp == NULL)
   {
      fprintf (stderr,"Couldn't open file for reading\n");
      exit(1);
   }
   uint8_t *array= (uint8_t*)malloc(4*sizeof(uint8_t));
   int i;
   for (i = 0; i < 4; i++) {
  fscanf(fp, "%x", &array[i]);
   }
   fclose(fp);
return array;
}
```

•Capteur utilisé :

Le capteur DHT11 est capable de mesurer des températures de 0 à +50°C avec une précision de +/- 2°C et des taux d'humidité relative de 20 à 80% avec une précision de +/- 5%. Le capteur a la particularité de communiquer avec le microcontrôleur via une unique broche d'entrée / sortie qui sera le pin 23.

Nous utiliserons ce code pour récupérer les données de la température et les écrire sur

le fichier qui sera lu par le code cité ci-dessus.
Nous avons relié la broche VCC du capteur à la broche 3,3V, la broche OUTPUT du capteur à la broche 23, le GND du capteur à une broche GPIO GND

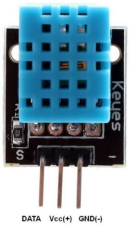

FIGURE 3.9 – DHT11

Nous avons utilisé la librairie Adafruit_DHT afin de lire à partir du capteur, puis transformer la valeur float lue en format IEE754 à l'aide des fonctions décrites ci-dessous. On écrit à chaque itération les nouvelles valeurs sur le fichier qui sera lu par l'application.

Nous avons donc importé les librairies nécessaires

```
import Adafruit_DHT
import time
```

Cette fonction nous permettra de transformer n'importe quelle type en format IEEE754.

```
def IEEE754(n) :
```

On récupère la température et l'humidité en utilisant la fonction disponible dans la libraire.

```
    humidity, temperature = Adafruit_DHT.read_retry(sensor, DHT11_pin)
```

Puis nous écrivons la température à chaque itération

```
 myfile.write(str(IEEE754(temperature)[x]) + '\n')
```

L'éxecution du code python doit se faire de façon à minimiser les efforts du CPU :

```
sudo nice -n 19 python main.py
```

"nice" est une commande disponible sur le système d'exploitation Linux. Cette commande pointe directement vers un point d'entrée du kernel portant le même nom, elle permet de changer le niveau de priorité d'un processus déterminé. La priorité la plus élevée correspond à un niveau de -20, tandis que la plus basse correspond à +19.

Dans notre cas nous donnons la priorité la plus basse pour permettre à notre application de respecter les contraintes de temps réel.
Les données dans le fichier seront sous cette forme :

FIGURE 3.10 – Valeur du capteur sous format IEEE754 dans le fichier

La communication cyclique se fait comme suit :
Voici le prototype de la fonction qui nous permettra de gérer la communication cyclique :

```
static void app_handle_cyclic_data(
  pnet_t * net,
  const app_data_t * p_appdata,
  uint8_t data_ctr,
  uint8_t * p_inputdata,
  uint16_t inputdata_size)
```

Nous devons en premier lieu récupérer la valeur du capteur et l'assigner au paramètre p_inputdata à l'aide de la fonction décrite plus haut, elle nous permettra de récupérer les données sous forme de vecteur et il nous sera plus facile de le transformer en float plus tard.

```
   p_inputdata = read_value_sensor("sensor.txt");

```

Cette fonction a déjà été cité dans l'API, elle nous permet d'envoyer les datas de l'IO device à l'IO controller ainsi que le IOPS (Provider status)

```
          (void)pnet_input_set_data_and_iops (
             net,
             APP_API,
             slot,
             p_subslot->subslot_nbr,
             p_inputdata,
             inputdata_size,
```

```
            iops);

        (void)pnet_input_get_iocs (
            net,
            APP_API,
            slot,
            p_subslot->subslot_nbr,
            &inputdata_iocs);

        pnet_output_get_data_and_iops (
            net,
            APP_API,
            slot,
            p_subslot->subslot_nbr,
            &outputdata_is_updated,
            outputdata,
            &outputdata_length,
            &outputdata_iops);
```

Toutes les nomitations output/input seront désignées par rapport à l'IO controller, c'est pour cela que les données envoyées de l'IO device à l'IO controller sont appelées input.

3.6.6 Implémentation de la communication acyclique

L'implémentation se fait grâce à une fonction de rappel (callbacks) qui est une fonction passée dans une autre fonction en tant qu'argument,et qui est ensuite invoquée à l'intérieur de la fonction externe pour accomplir une sorte de routine ou d'action.

```
static int app_write_ind

static int app_read_ind
```

Comme nous pouvons le constater, ces fonctions sont rappelées autre part dans le code sous forme de callbacks.

```
int app_pnet_cfg_init_default (pnet_cfg_t * stack_config)
{
...
```

```
  stack_config->read_cb = app_read_ind;
  stack_config->write_cb = app_write_ind;
...
}
```

3.6.7 Implémentation des alarmes

Voici le prototype de la fonction qui s'occupera des alarmes.

```
static void app_handle_send_alarm(
  pnet_t * net,
  uint32_t arep,
  bool * p_alarm_allowed,
  const app_data_t * p_appdata,
  uint8_t * alarm_payload)
```

Elle utilise cette fonction qui a été décrite dans la partie API :

```
        pnet_alarm_send_process_alarm (
           net,
           arep,
           APP_API,
           slot,
           p_subslot->subslot_nbr,
           APP_ALARM_USI,
           APP_ALARM_PAYLOAD_SIZE,
           alarm_payload);
```

3.6.8 Application prototype

Notre application a la spécificité de transmettre les données lues d'un capteur en temps réel sur le canal cyclique, paramétrer le temps entre chaque lecture du capteur et un seuil sur le canal acyclique, qui en cas de dépassement, engendrera une alarme qui sera lancée en temps réel sur le canal adéquat.
Pour pouvoir mettre en place la condition pour le lancement de l'alarme avec le seuil reçu sous forme de paramètre acyclique :
Nous avons dû effectuer des opérations de manipulation de données pour convertir entre les différents types de données.

Pour convertir les données lues du fichier en float lisible par notre système, nous avons utilisé une union qui, à l'image d'une structure, est un regroupement d'objets de

type différents, occupant le même espace de mémoire.

```
union u {
uint8_t x[4];
float val;
};

```

Après avoir fait l'instanciation de notre union, on lit notre fichier puis on inverse l'ordre des octets, on passe d'une disposition BE (Big endian) à une disposition LE (Little endian).

```
union u untest;

uint8_t* buffer = read_value_sensor("sensor.txt");
untest.x[3] = buffer[0];
untest.x[2] = buffer[1];
untest.x[1] = buffer[2];
untest.x[0] = buffer[3];

```

On effectue aussi la conversion de Big endian à Little endian avec le paramètre acyclique qui représente le seuil mais cette fois-ci à l'aide d'une fonction qui nous permet de réarranger l'ordre de nos données sachant que le paramètre est de type float.

```
threshold = ReverseFloat(p_appdata->app_param_1);
```

Voici la fonction utilisée pour inverser une donnée float de type IEEE754 d'une disposition Big endian à une disposition Litte endian ou vice versa.

```

float ReverseFloat( const float inFloat )
{
   float retVal;
   char *floatToConvert = ( char* ) & inFloat;
   char *returnFloat = ( char* ) & retVal;
   returnFloat[0] = floatToConvert[3];
   returnFloat[1] = floatToConvert[2];
   returnFloat[2] = floatToConvert[1];
   returnFloat[3] = floatToConvert[0];
   return retVal;
}
```

Puis on utilise le paramètre reçu comme condition pour lancer l'alarme en cas de dépassement du seuil, on utilise aussi une variable pour contrôler la machine d'état et

limiter le lancement de l'alarme à 4 fois, assez pour exécuter les 4 états cités plus tôt d'une alarme.

```
            if ( untest.val> threshold && k==false)
        {
          for(int i=0;i<4;i++) {
          app_handle_send_alarm (
             net,
             p_appdata->main_api.arep,
             &p_appdata->alarm_allowed,
             p_appdata,
             alarm_payload);
          }
          k=true;
        }

```

Nous avons utilisé toutes ces fonctions dans une boucle qui les exécutera selon le flag indiquant l'état actuel, qui varie selon les événements manipulés par l'OS à l'aide des fonctions dédiées à cela.

Exemple :

Pour sortir de l'état de l'alarme :

```
os_event_clr (p_appdata->main_events, APP_EVENT_ALARM)

```

3.6.9 Conclusion

Dans ce chapitre, nous avons fait l'implémentation de notre IO device en effectuant une analyse par rapport au hardware et système d'exploitation à choisir tout en prenant en compte les besoins de l'application.

Nous avons ensuite fait en sorte que notre système respecte les contraintes en temps réel de notre application en utilisant plusieurs méthodes afin d'accomplir ceci.

Après une étude approfondie de la bibliothèque p-net et de son implémentation, nous avons programmé une application en utilisant son API pour concevoir un IO device fonctionnel qui sera testé dans le prochain chapitre.

Chapitre 4

Test et résultats

4.1 Introduction

Pour tester l'IO device, nous avons simulé un PLC à l'aide d'une autre raspberry pi, nous avons utilisé codesys pour pouvoir faire cela, après avoir reproduit ce qu'aurait fait un automaticien, c'est à dire, ajouter le GSD file du device dans codesys (ou le logiciel consacré au PLC) puis ajouter notre device dans notre topologie et régler tout les paramètres nécessaires (IP ...) puis connecter l'IO device et l'IO controller dans le même réseau,nous allons ensuite recueillir les trames et packets à l'aide de wireshark et contrôler l'efficacité de nos méthodes pour le respect des contraintes en temps réel sur notre système.

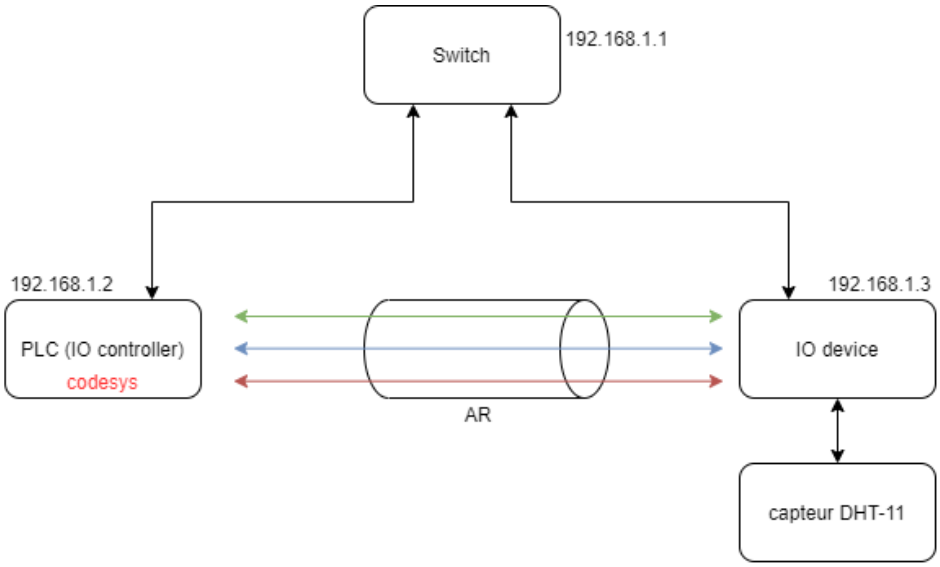

FIGURE 4.1 – Prototype du setup

4.2 Outils utilisés

4.2.1 Codesys

CODESYS est un environnement de développement pour des automates programmables industriels (API) selon le standard CEI 61131-3 pour le développement d'applications dans l'automation industrielle.

Il nous permettra de simuler le comportement d'un PLC sur une raspberry pi, afin de lire les données transmises en temps réel sur le canal cyclique, lire et modifier les paramètres acycliques, et nous informer dans le cas du déclenchement d'une alarme.

Après l'installation du logiciel et le téléchargement de tout les plugins nécessaires au déploiement sur raspberry pi, nous allons ensuite insérer les paramètres spécifiques à notre système, c'est à dire, le **username** et **mot de passe** du root, l'**adresse IP**, puis installer le package et lancer le runtime.

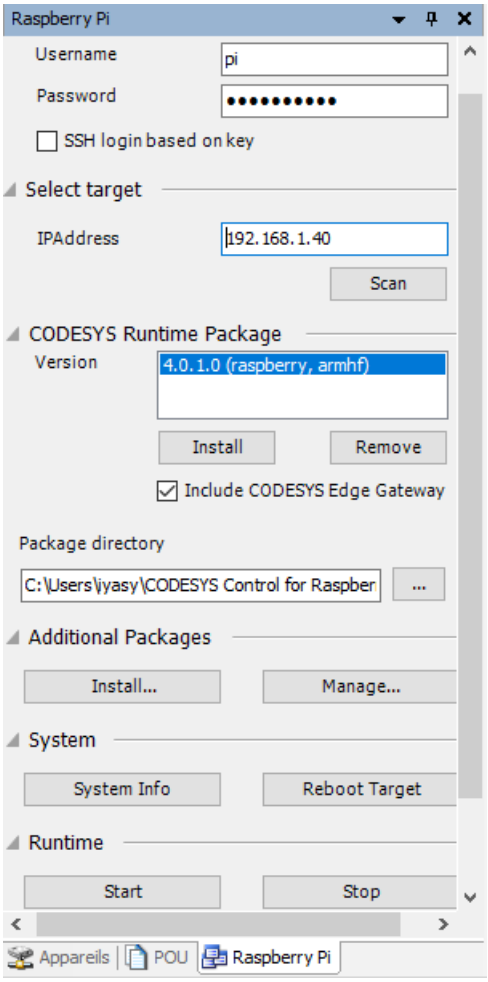

FIGURE 4.2 – Configuration de la raspberry qui simule le PLC

Ensuite, nous avons ajouté le GSD file de notre IO device au projet, pour cela nous nous sommes rendus sur l'onglet outils puis Référentiel d'appareils, "installer", et on choisit le GSDML file confectionné par nos soin.

La prochaine étape sera d'ajouter notre IO device dans l'onglet appareil, dans **Device (CODESYS Control for Raspberry pi SL)** , on appuie sur Ajouter un appareil, Adapteur ethernet, Ethernet.

Après un clic droit sur ethernet, on choisit l'option Ajouter un appareil, Profinet IO master, puis PN-Controller.

On reproduit la même chose sur PN-Controller, on ajoute "P-Net Sample App", puis DIO 8xLogicLevel.

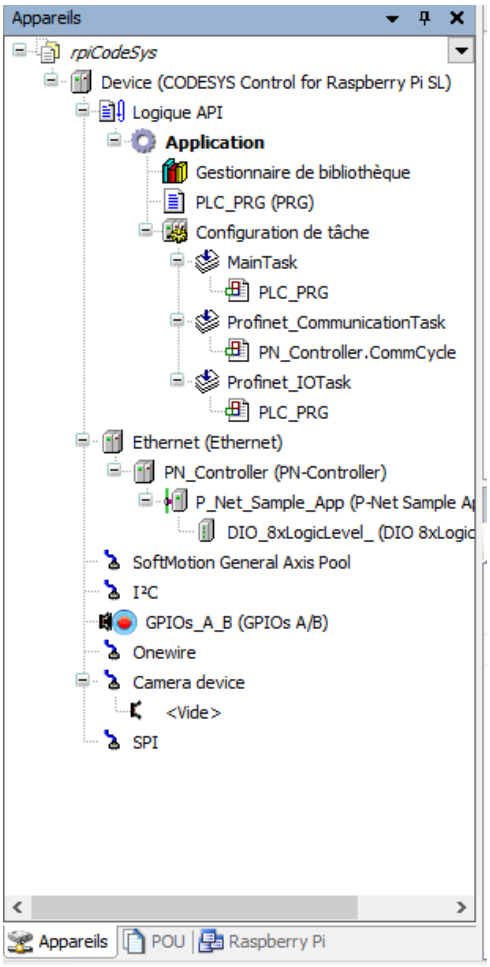

FIGURE 4.3 – Interface du logiciel après ajout des composants

Sur l'onglet Ethernet, on ajuste l'adresse IP, le masque de sous-réseau et la passerelle par défaut de notre interface réseau "eth0" qui définit les paramètres de notre **IO device**.

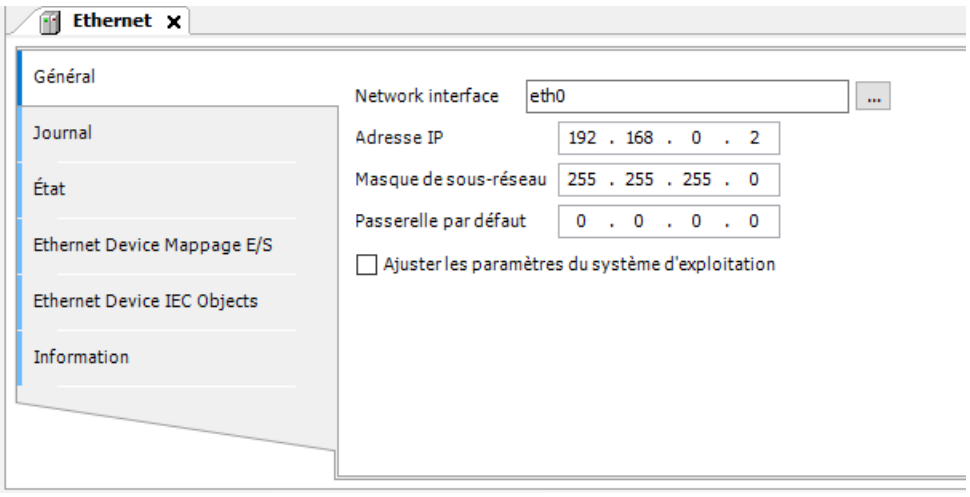

FIGURE 4.4 – Paramètrage de l'interface ethernet de la raspberry simulé en PLC

Sur l'onglet PN-controller, on ajuste la plage d'adresses IP de sorte à couvrir tous les futurs IO devices.

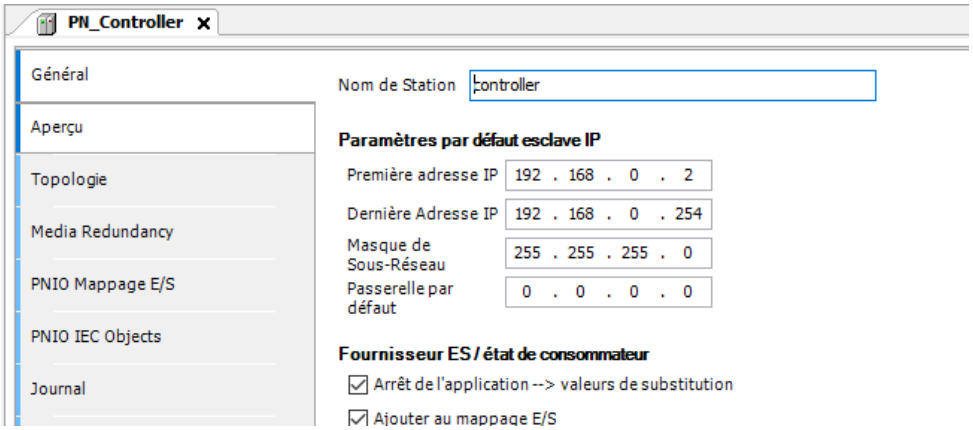

FIGURE 4.5 – Paramètre de la plage d'adresse des IO devices

Sur l'onglet PN_Net_Sample_App, on paramètre l'adresse IP, le masque de sous-réseau et la passerelle par défaut de l'IO device.

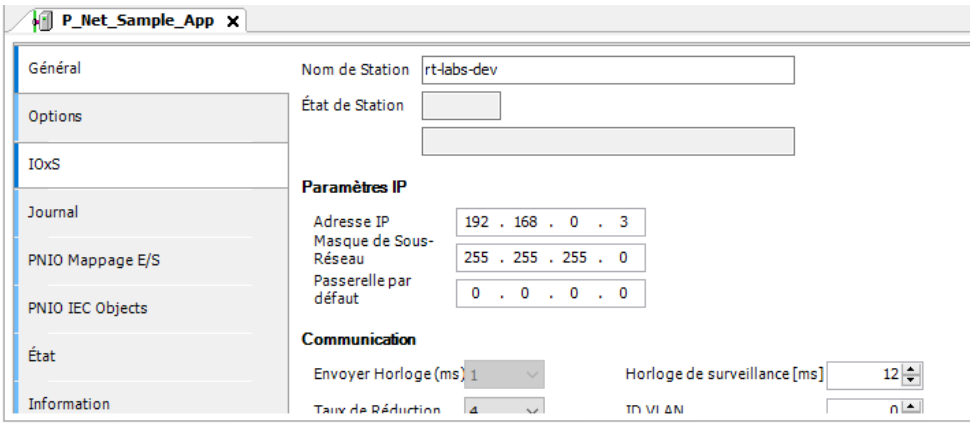

FIGURE 4.6 – Paramétrage de l'IO device à travers le logiciel

Il ne reste plus qu'à se connecter et appuyer sur Démarrer et le processus sera lancé.

4.2.2 Wireshark

Wireshark est un logiciel d'analyse réseau (sniffer) qui permet de visualiser l'ensemble des données transitant sur la machine qui l'exécute, et d'obtenir des informations sur les protocoles applicatifs utilisés.[16]

Le filtre qui nous permet de ne visualiser que les paquets de type profinet se présente sous cette forme : pn_io

4.3 Test de l'application

4.3.1 Test du lancement de l'application

1-Le PLC envoie un message de diffusion DCP (broadcast) et tous les IO devices du sous-réseau répondent en indiquant leur adresse MAC.

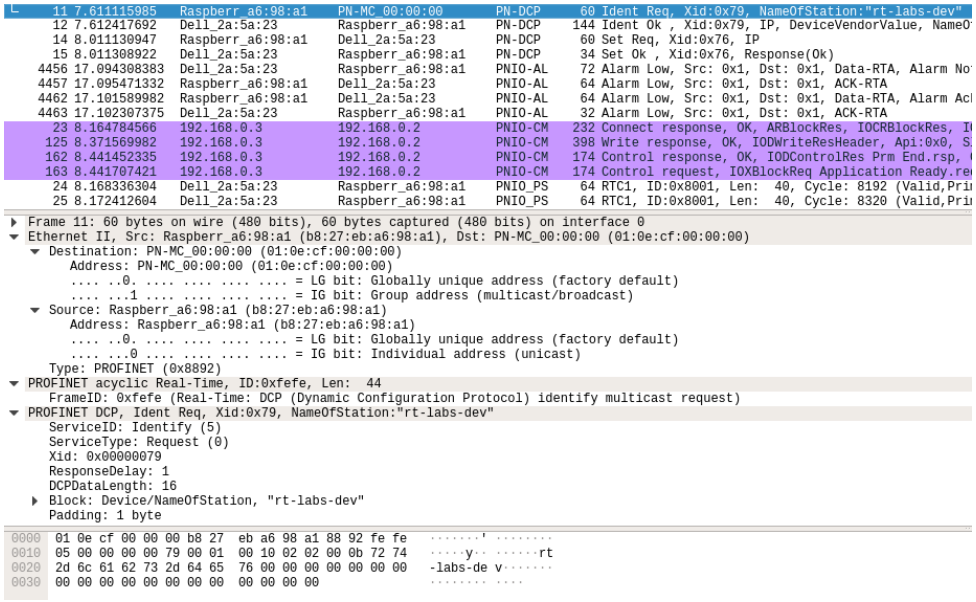

FIGURE 4.7 – Première trame DCP broadcast

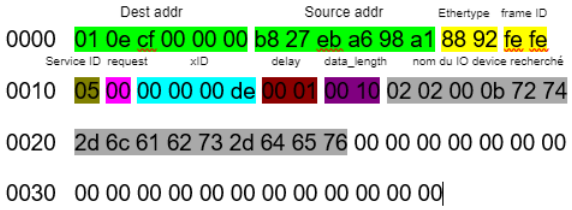

FIGURE 4.8 – Première trame DCP broadcast en octets

Service : Requête d'identification qui est décrite par un Service ID égal à 5 et un type de service égal à 0.

Le bloc gris désigne le nom du device recherché et nous permet de trouver notre IO device qui est identifié par son nom seulement.

Nous remarquons aussi que les trames DCP ne contiennent pas de tag VLAN et ne sont donc pas concernées par les priorités, elles possédent directement un ethertype 0x8892 correspondant au protocole PROFINET, ce sera le cas pour toutes les trames DCP.

2-l'IO device confirme son nom :

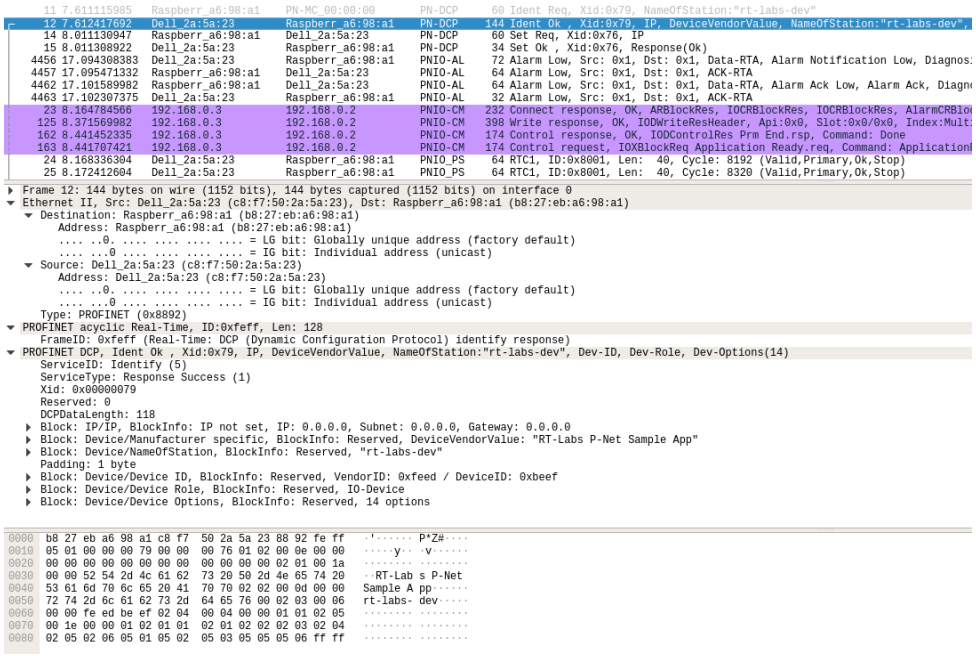

FIGURE 4.9 – Deuxième trame DCP confirmant le nom de l'IO device

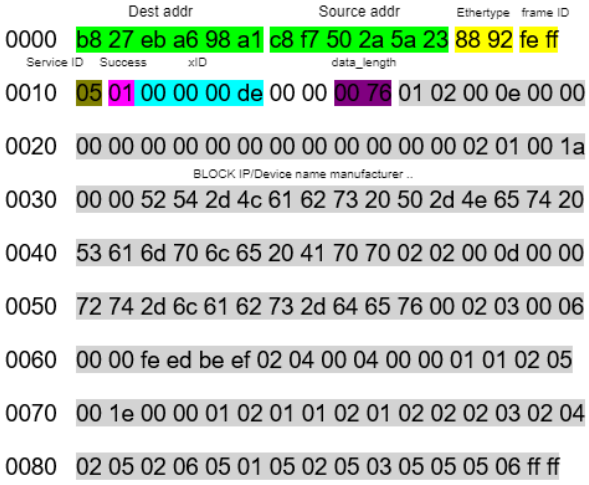

FIGURE 4.10 – Deuxième trame DCP en octets

Service : Réponse d'identification qui est décrite par un Service ID égal à 5 et un type de service égal à 1.

Le bloc gris désigne la configuration actuelle de l'IO device ainsi que toutes les informations additionnelles de celui-ci .

3-L'Automate envoie un message DCP à l'IO device en utilisant son adresse MAC, contenant l'adresse IP qu'il doit utiliser.

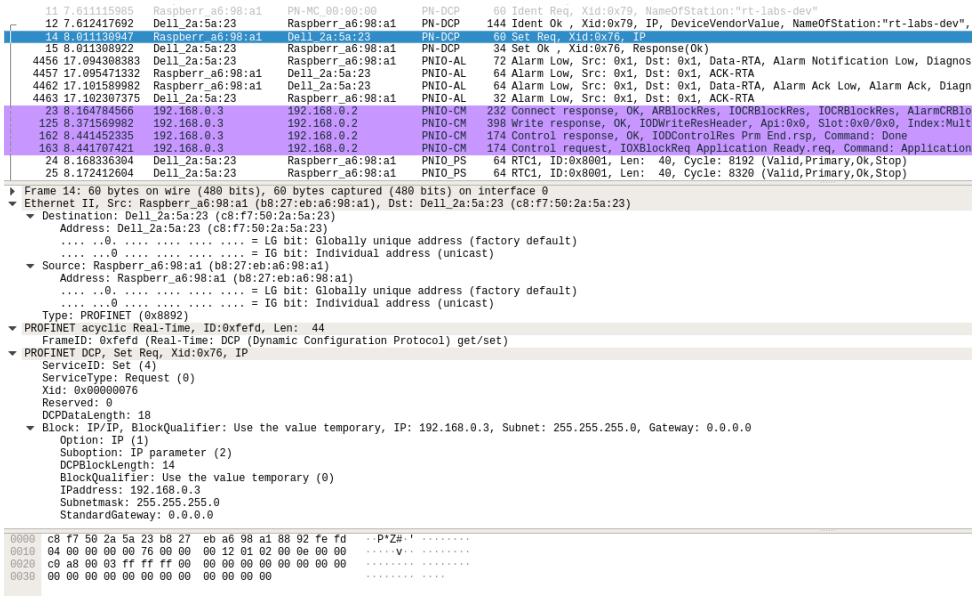

FIGURE 4.11 – Troisième trame DCP qui montre l'envoi de l'adresse IP

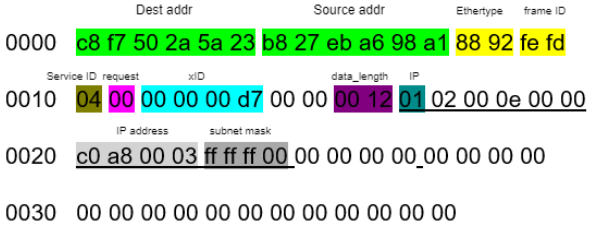

FIGURE 4.12 – Troisième trame DCP en octets

Service : Requête de réglage qui est décrite par un Service ID égal à 4 et un type de service égal à 0.

Le bloc gris désigne l'IP, le masque de sous-réseau ainsi que la passerelle par défaut qui sont envoyées par le PLC à l'IO device.

4-l'IO device confirme que l'adresse a été reçue et configurée :

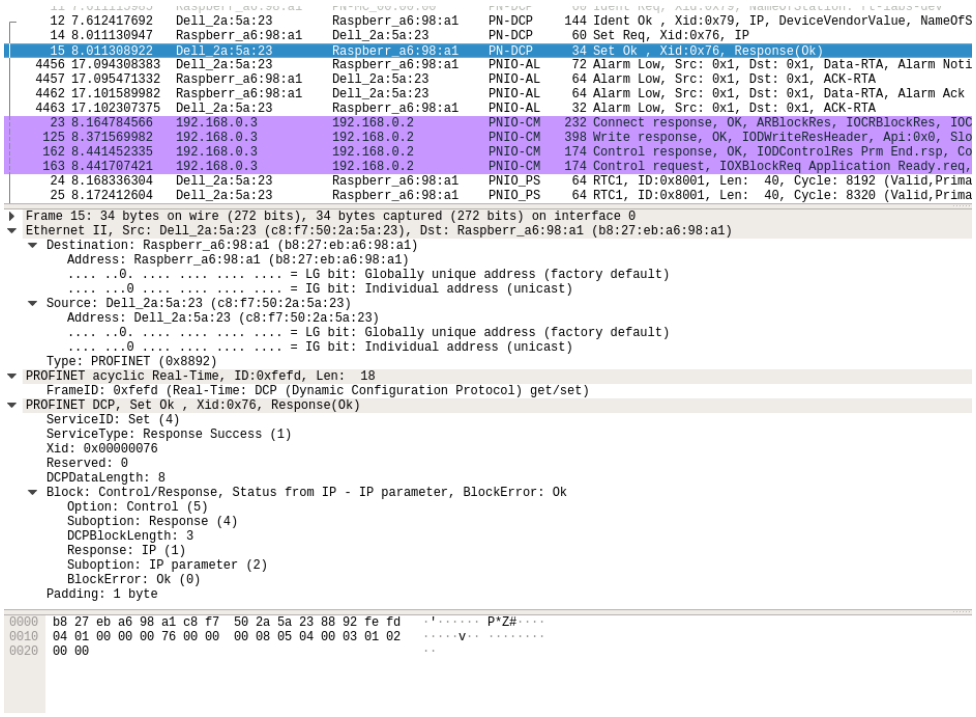

FIGURE 4.13 – Dernière trame DCP confirmant la réception des paramètres par l'IO device

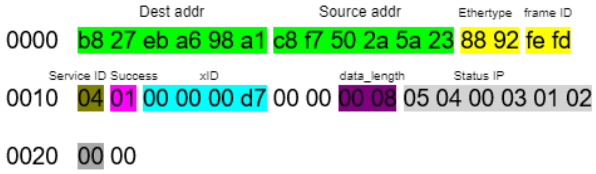

FIGURE 4.14 – Dernière trame DCP en octets

Service : Réponse de réglage qui est décrite par un Service ID égal à 4 et un type de service égal à 1.

- LLDP :

Les trames LLDP sont utiles pour détecter à quel voisin, le dispositif est connecté dans le réseau.

Elles sont envoyées par un dispositif Profinet toutes les 5 secondes, pour indiquer l'adresse IP, etc.

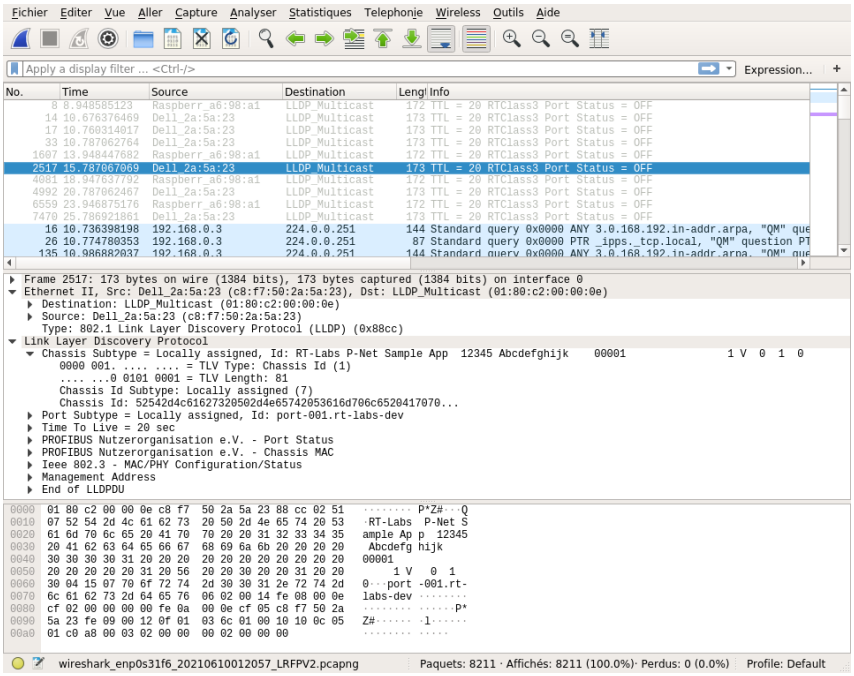

FIGURE 4.15 – Trame LLDP de l'IO device

4.3.2 Test communication des données cycliques

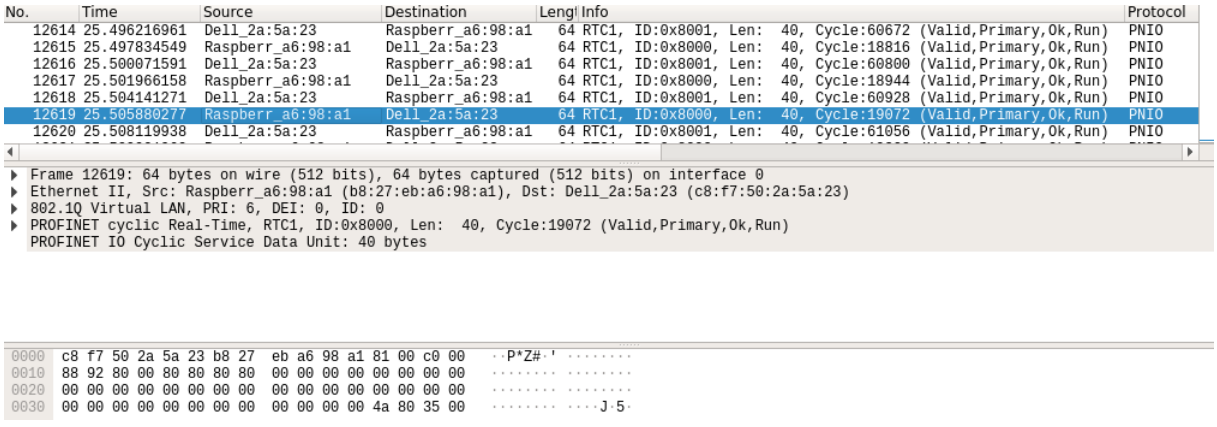

FIGURE 4.16 – Trame donnée cyclique de l'IO controller à l'IO device

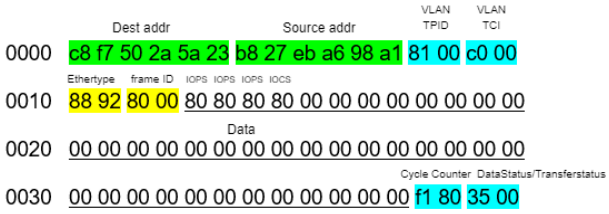

FIGURE 4.17 – Trame donnée cyclique de l'IO controller à l'IO device en hexa

Détails des 40 octets de data envoyés de l'IO controller à l'IO device :

byte 1 : IOPS du slot 0, subslot 1.

byte 2 : IOPS du slot 0, subslot 0x8000.

byte 3 : IOPS du slot 0, subslot 0x8001.

byte 4 : IO data du slot 1, subslot 1.

byte 5 : IOPS du slot 1, subslot 1.

byte 6 : IOCS du slot 1, subslot 1.

(puis 31 bytes de padding à 0)

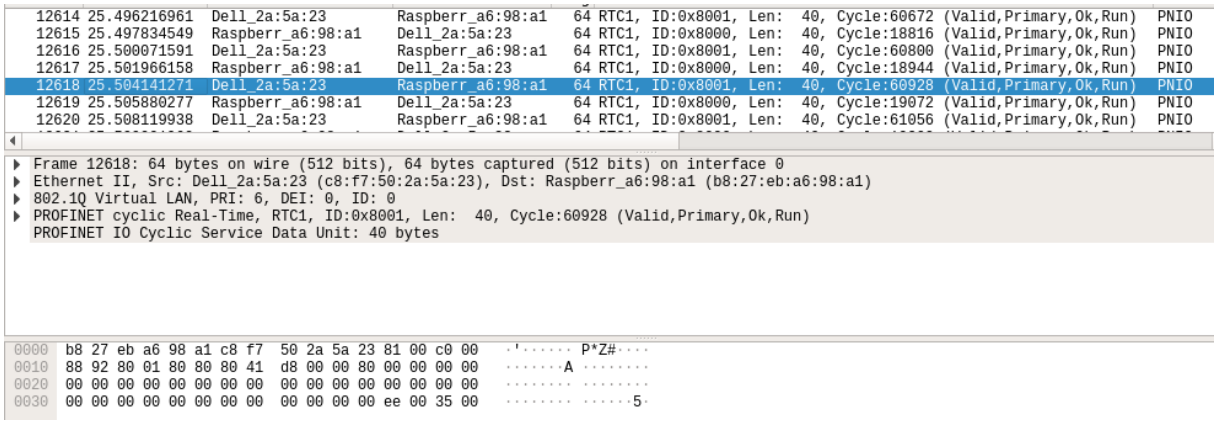

FIGURE 4.18 – Trame donnée cyclique de l'IO device à l'IO controller

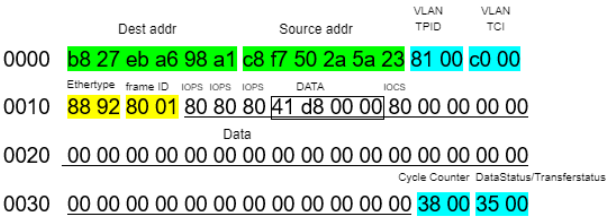

FIGURE 4.19 – Trame donnée cyclique de l'IO device à l'IO controller en hexa

Détails des 40 octets de data envoyés de l'IO device à l'IO controller :

byte 1 :IOPS (Provider Status) du slot 0, subslot 1.

byte 2 :IOPS (Provider Status) du slot 0, subslot 0x8000.

byte 3 :IOPS (Provider Status) du slot 0, subslot 0x8001.

byte 4-8 : IO data du slot 1, subslot 1 (4 bytes data).

byte 9 :IOPS (Provider Status) du slot 1, subslot 1.

byte 10 :IOCS (Consumer Status) du slot 1, subslot 1.

(puis 31 bytes de padding à 0)

Dans les trames cycliques, nous avons le tag ethertype VLAN 0x8100 qui est suivi par le Tag Control Information qui contient la priorité de la trame (6), le format, et l'ID qui sont à 0 suivi par l'ethertype 0x8892 représentant PROFINET.

4.3.3 Test communication des données acycliques

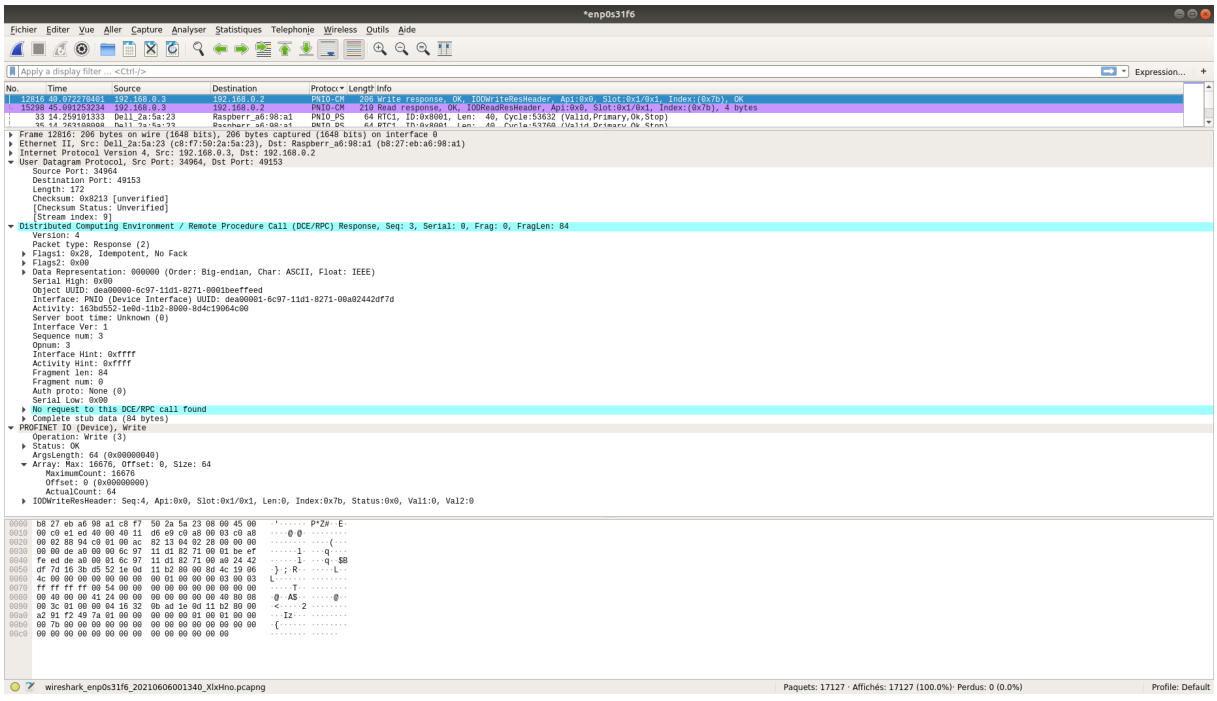

FIGURE 4.20 – Write response (écriture)

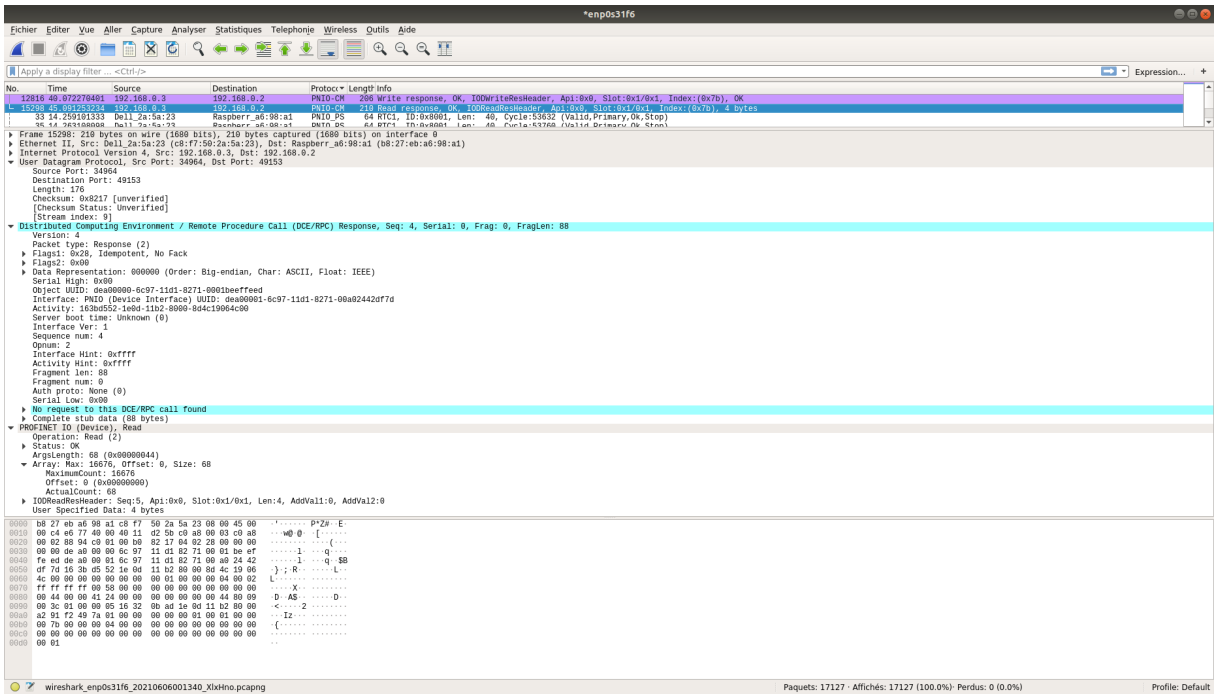

FIGURE 4.21 – Read response (lecture)

4.3.4 Test des alarmes

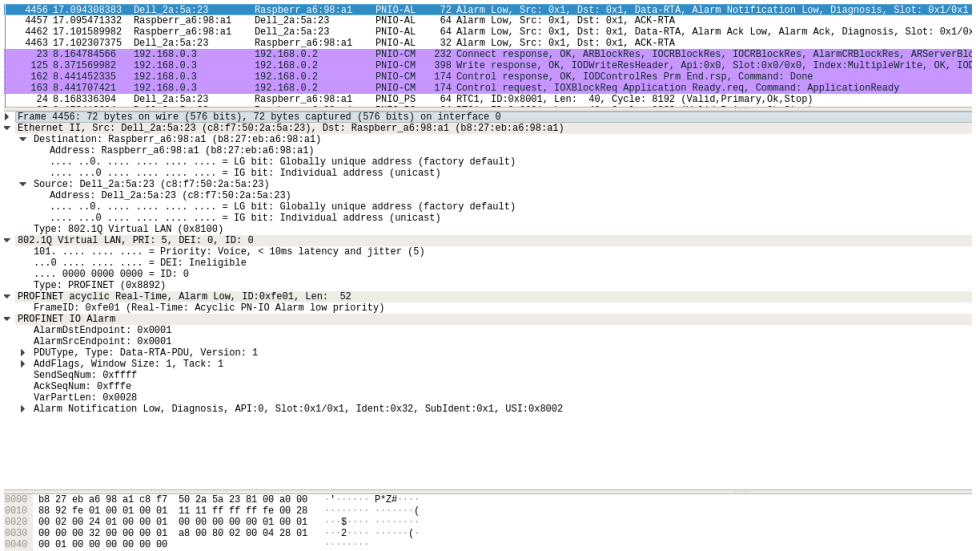

FIGURE 4.22 – Trame alarme notification

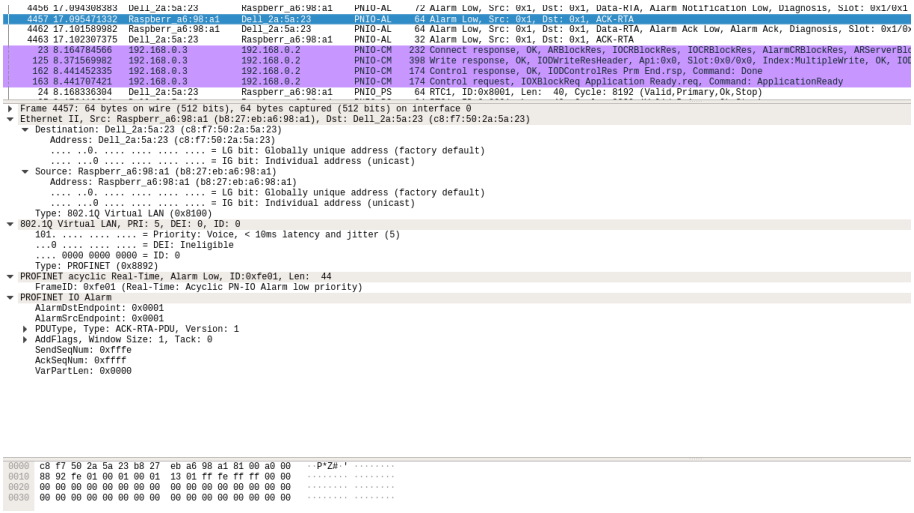

FIGURE 4.23 – Trame alarme acknowledgement

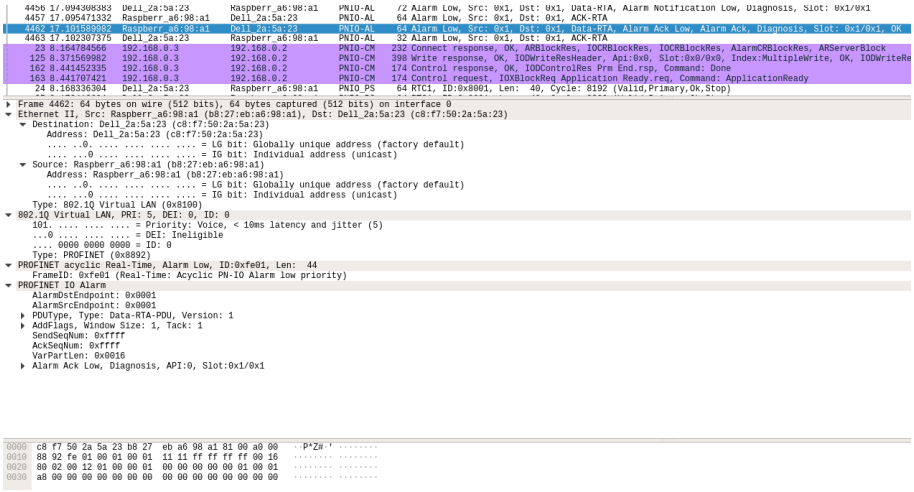

FIGURE 4.24 – Trame alarme réponse du PLC

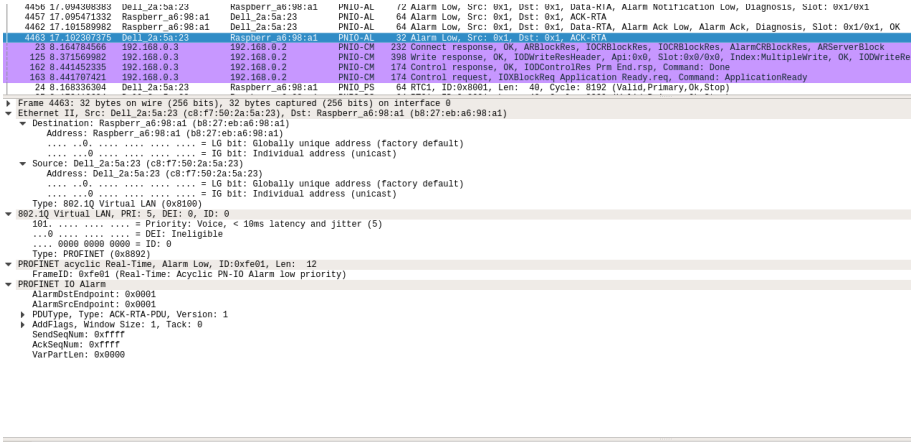

FIGURE 4.25 – Trame alarme acknowledgement

4.4 Performance temps réel

Notre application nous impose des contraintes de temps réel qui doivent être respectées. Pour pouvoir accomplir cela, nous avons utilisé plusieurs méthodes qui nous ont permis de réduire le temps de réponse de notre application. La variation de la latence dans le temps est appelée jitter. Un jitter élevé signifie que les délais sont fortement variables, ce qui perturbe les protocoles en temps réel. Nous avons donc effectué des tests sur notre système linux pour comparer l'impact de nos méthodes sur la latence et le jitter de notre application.

Cyclictest est le plus souvent utilisé pour le benchmarking des systèmes RT. C'est l'un des outils les plus fréquemment utilisés pour évaluer les performances relatives des systèmes temps réel.

Il mesure de manière précise et répétée la différence entre la transition des états des threads afin de fournir des informations sur la variation de latences du système. Il peut mesurer les jitters causées par le matériel, le firmware et le système d'exploitation.

4.4.1 Performance avec un noyau standard

•La figure 4.26 représente la latence des différents coeurs du processeur sur un échantillon de thread qui nous permet de juger les performances de notre système avec un noyaux linux standard, et sans l'isolation de l'application sur un coeur du processeur.

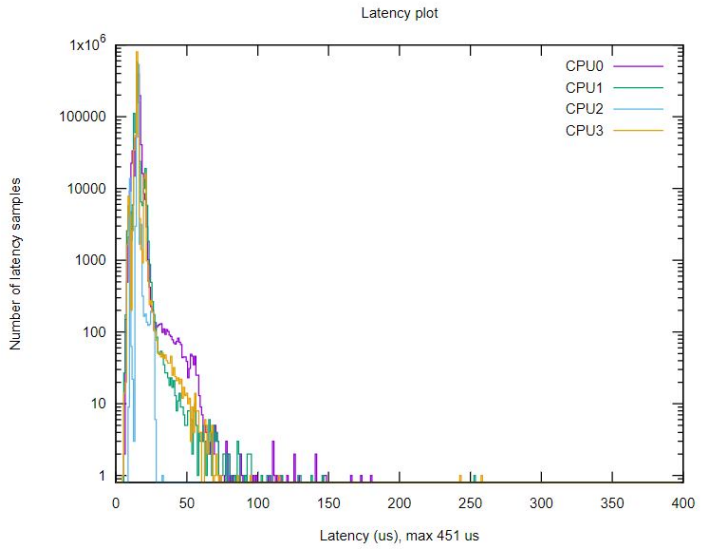

FIGURE 4.26 – Latence avec noyau standard et sans taskset

```
sudo sh mklatencyplot
```

Il y'a une forte variation de latences, avec une latence maximale de 451 us, qui peut causer l'arrêt de notre application.

•La figure 4.27 représente la latence des différents coeurs du processeur sur un échantillon de thread qui nous permet de juger les performances de notre système avec un noyaux linux standard, et avec l'isolation de l'application sur le coeur 2 du processeur à l'aide de la manipulation citée dans le chapitre 3.

```
sudo taskset -c 2 sh mklatencyplot
```

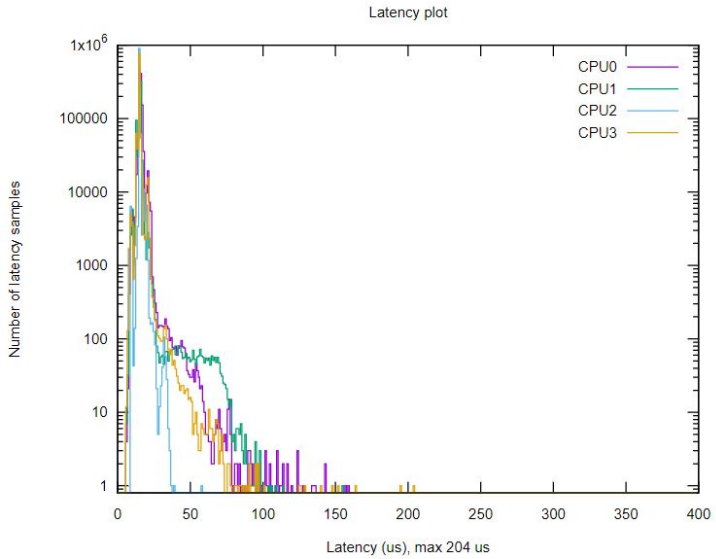

FIGURE 4.27 – Latence avec noyau standard et taskset

Nous observons la diminution de la latence comparée à la figure 4.26 avec une latence maximale de 204us.

4.4.2 Performance avec noyau temps réel

•La figure 4.28 représente la latence des différents coeurs du processeur sur un échantillon de thread qui nous permet de juger les performances de notre système avec un noyau linux temps réel RT_PREEMPT, implémenté à l'aide de la manipulation citée dans le chapitre 3 et sans l'isolation de l'application sur le coeur 2.

```
sudo sh mklatencyplot
```

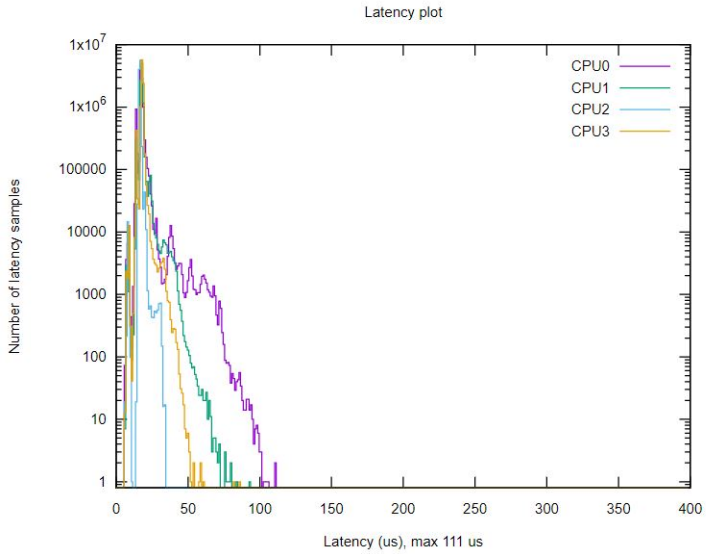

FIGURE 4.28 – Latence avec noyau temps réel et sans taskset

On remarque beaucoup moins de variations par rapport aux tests effectués avec un noyau standard. La latence maximale est de 111 us.

•La figure 4.29 représente la latence des différents coeurs du processeur sur un échantillon de thread qui nous permet de juger les performances de notre système avec un noyau linux temps réel RT_PREEMPT, et avec l'isolation de l'application sur le coeur 2 du processeur à l'aide des manipulations citée dans le chapitre 3.

```
sudo taskset -c 2 sh mklatencyplot
```

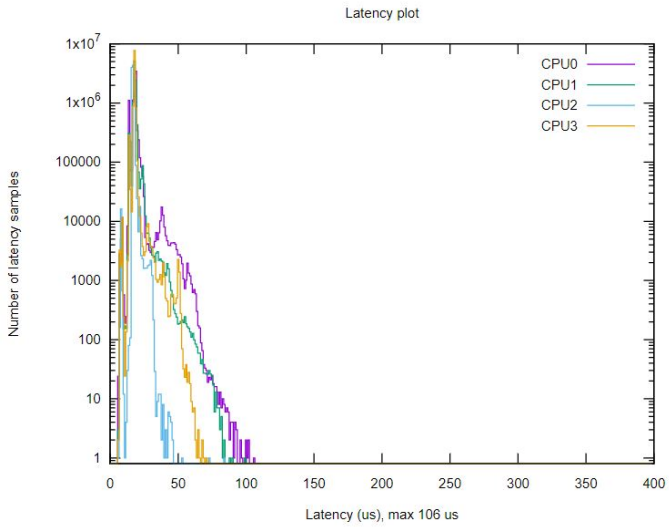

FIGURE 4.29 – Latence avec noyau temps réel et taskset

Il s'agit de la combinaison de méthodes qui nous a fourni la variation de latences la plus basse ainsi qu'une latence maximale de 106us.

4.4.3 Conclusion

Comme nous pouvons l'observer sur les graphes résultants de nos tests, la combinaison des méthodes choisies, c'est à dire, l'utilisation d'un noyau temps réel ainsi que l'isolation de l'application sur un coeur séparé minimise la variation de latences et nous permet donc de respecter les contraintes de temps réel imposées par le protocole.

Conclusion générale

Ce travail contribue à l'intégration du protocole de communication PROFINET sur système embarqué. Nous avons eu l'occasion de voir l'importance des bus de communication dans un environnement industriel et des systèmes en temps réel.
La compréhension de l'architecture industrielle et de son organisation nous a permis d'adhérer à l'idée de la transition de l'analogique au numérique et de l'intérêt des bus de communication basés sur ethernet.
Ensuite, nous avons découvert et assimilé le protocole PROFINET en montrant toutes ses fonctionnalités et en faisant une analyse approfondie de chaque partie caractérisant ce protocole.
Nous avons, à l'aide de solutions open source, implémenté et conçu une application spécifique exploitant toutes les fonctionnalités de PROFINET. Nous avons aussi réalisé une maquette fonctionnelle pour les tests.
Notre application exige un respect des contraintes de temps réel strict, nous avons donc adapté notre système d'exploitation linux afin qu'il réponde aux contraintes de l'application en utilisant plusieurs méthodes citées dans le chapitre 3.
Enfin nous avons assuré le bon fonctionnement de notre application implémentée en captant toutes les données transférées et contrôler l'efficacité de nos méthodes pour le respect des contraintes en temps réel sur notre système.

Il convient de souligner les limites causées par la librairie et le hardware pour réaliser un IO device de classe supérieure. Néanmoins, notre travail est largement suffisant pour l'application demandée.

Par rapport aux perspectives de recherches futures, le projet pourrait se porter sur la conception d'un IO device sur STM32 en utilisant un RTOS ce qui permettra un meilleur contrôle sur tous les aspects de l'application ainsi qu'un hardware développé sur mesure pour la réalisation d'un produit fini. On peut aussi partir sur une solution plus générale, qui consitera en la réalisation d'une image linux adaptée à notre application à l'aide de yocto et d'y intégrer différents bus de communication.

Bibliographie

[1] C. NIEL Éric, *Maîtrise des risques et sûreté de fonctionnement des systèmes de production*. Éditeur : Hermes Science Publications (25 février 2002).

[2] "What is a distributed control system (dcs)," [consulté le 3 mai 2021]. Disponible sur : <https://www.electricaltechnology.org/2016/08/distributed-control-system-dcs.html>.

[3] N. Ayllon, "Fieldbus vs 4-20 ma," [consulté le 15 mai 2021]. Disponible sur : <https://us.profinet.com/fieldbus-vs-4-20-ma/>.

[4] D. Abdelhamid, "Cours réseaux locaux industriels," 2010 Disponible sur <http://www.abdelhamid-djeffal.net/web_documents/polycope_rli_10.pdf>.

[5] AGILICOM, [consulté le 2 avril 2021]. Disponible sur : <https://www.agilicom.fr/tutorial-PROFINET-IO.html>.

[6] "Profinet théorie et pratique [consulté le 20 avril 2021]. disponible sur : <https://www.profibus.com/index.php?eID=dumpFile&t=f&f=35519&token=225153add677d46a889856bb7e16f7e580ced7d4>."

[7] R. Pigan and M. Metter, *Automating with PROFINET, Industrial Communication based on Industrial Ethernet*. Publisher : Publicis (December 3, 2008).

[8] "p-net profinet device stack docs [consulté le 6 avril 2021]. disponible sur : <https://rt-labs.com/docs/p-net/_copied/README.html>."

[9] "Technical specification : Raspberry pi 3 model b," [consulté le 2 juin 2021]. Disponible sur : <https://www.terraelectronica.ru/pdf/show?pdf_file=%252Fds%252Fpdf%252FT%252FTechicRP3.pdf>.

[10] P. V. Hung, "Linux embarqué et système embarqué," Hanoï, 15 Juillet 2005. [consulté le 24 Juin 2021]. Disponible sur : <https://kadionik.vvv.enseirb-matmeca.fr/embedded/tipe-pham_viet_hung.pdf>.

[11] D. Abbott, *Linux for Embedded and Real-time Applications, Fourth Edition*. Publisher : Newnes, 2017.

[12] M. Riva, Real Time System - Preempt-RT Patching Tutorial [consulté le 13 Juin 2021]. Disponible sur <https://lemariva.com/blog/2018/07/raspberry-pi-preempt-rt-patching-tutorial-for-kernel-4-14-y>.

[13] J. Fix, "Tutoriel cmake , [consulté le 28 mai 2021]. disponible sur <http://sirien.metz.supelec.fr/depot/SIR/TutorielCMake/index.html#principe>."

[14] geeksforgeeks, "Socket programming in c/c++," [consulté le 2 mai 2021]. Disponible sur : <https://www.geeksforgeeks.org/socket-programming-cc/>.

[15] "Capteur de température et humidité dht11 avec raspberry pi, [consulté le 2 juin 2021]. disponible sur <https://www.freva.com/fr/2019/05/22/capteur-de-temperature-et-humidite-dht11-avec-raspberry-pi/>."

[16] Hilscher, "Capturing profinet with wireshark," [consulté le 12 mai 2021]. Disponible sur <https://www.hilscher.com/>, DOC190402AN01EN.

[17] Open Source Automation Development Lab (OSADL), Cyclictest script [consulté le 2 Juin 2021]. Disponible sur <https://www.osadl.org/uploads/media/mklatencyplot.bash>.

[18] IEEE-754 : Aide Mémoire [consulté le 25 mai 2021]. Disponible sur <https://sifflez.org/lectures/archi-ord/AideMemoireIEEE754.pdf>.

Annexes

.1 Annexe A : Code application en C

- Code de la communication cyclique :

```
static void app_handle_cyclic_data (
  pnet_t * net,
  const app_data_t * p_appdata,
  uint8_t data_ctr,
  uint8_t * p_inputdata,
  uint16_t inputdata_size)
{
  uint16_t slot = 0;
  uint16_t subslot = 0;
  uint8_t inputdata_iocs = PNET_IOXS_BAD;  /* Consumer status from PLC */
  uint8_t outputdata_iops = PNET_IOXS_BAD; /* Producer status from PLC */
  uint8_t outputdata[APP_DATASIZE_OUTPUT];
  uint16_t outputdata_length = 0;
  uint8_t iops = PNET_IOXS_BAD;
  const app_subslot_t * p_subslot = NULL;
  bool outputdata_is_updated = false; /* Not used in this application */
  bool led_state = false;             /* LED for cyclic data */
  bool has_set_led_state = false;
  uint8_t output_parameter;
  /* Prepare input data (for sending to IO-controller) */

   p_inputdata = read_value_sensor("sensor.txt");

  for (slot = 0; slot < PNET_MAX_SLOTS; slot++)
  {
     for (subslot = 0; subslot < PNET_MAX_SUBSLOTS; subslot++)
     {
        iops = PNET_IOXS_BAD;
        p_subslot = &p_appdata->main_api.slots[slot].subslots[subslot];

        /* Set inputdata for all custom input modules, if any */
        if (p_subslot->plugged && app_subslot_is_input (p_subslot))
        {
           if (p_subslot->p_in_data != NULL)
           {
              iops = PNET_IOXS_GOOD;
           }
```

```
38
39          (void)pnet_input_set_data_and_iops (
40             net,
41             APP_API,
42             slot,
43             p_subslot->subslot_nbr,
44             p_inputdata,
45             inputdata_size,
46             iops);
47
48          (void)pnet_input_get_iocs (
49             net,
50             APP_API,
51             slot,
52             p_subslot->subslot_nbr,
53             &inputdata_iocs);
54
55          if (p_appdata->arguments.verbosity > 1)
56          {
57             if (inputdata_iocs == PNET_IOXS_BAD)
58             {
59                printf (
60                   "The controller reports IOCS_BAD for input slot %u\n",
61                   slot);
62             }
63             else if (inputdata_iocs != PNET_IOXS_GOOD)
64             {
65                printf (app_handle_cyclic_data,
66                   "The controller reports IOCS %u for input slot %u. Is
   it "
67                   "in STOP mode?\n",
68                   inputdata_iocs,
69                   slot);
70             }
71          }
72       }
73
74       /* Set outputdata for custom output modules, if any */
75       if (p_subslot->plugged && app_subslot_is_output (p_subslot))
76       {
```

```
77              outputdata_length = sizeof (outputdata);
78              pnet_output_get_data_and_iops (
79                 net,
80                 APP_API,
81                 slot,
82                 p_subslot->subslot_nbr,
83                 &outputdata_is_updated,
84                 outputdata,
85                 &outputdata_length,
86                 &outputdata_iops);
87           }
88        }
89     }
90  }
```

- Code de la communication acyclique :

```
1  static int app_write_ind (
2    pnet_t * net,
3    void * arg,
4    uint32_t arep,
5    uint32_t api,
6    uint16_t slot,
7    uint16_t subslot,
8    uint16_t idx,
9    uint16_t sequence_number,
10   uint16_t write_length,
11   const uint8_t * p_write_data,
12   pnet_result_t * p_result)
13
```

```
1  static int app_read_ind (
2    pnet_t * net,
3    void * arg,
4    uint32_t arep,
5    uint32_t api,
6    uint16_t slot,
7    uint16_t subslot,
8    uint16_t idx,
9    uint16_t sequence_number,
10   uint8_t ** pp_read_data,
11   uint16_t * p_read_length,
```

```
12   pnet_result_t * p_result)
```

- Code pour l'implémentation des alarmes :

```
1  static void app_handle_send_alarm (
2    pnet_t * net,
3    uint32_t arep,
4    bool * p_alarm_allowed,
5    const app_data_t * p_appdata,
6    uint8_t * alarm_payload)
7  {
8    static app_demo_state_t state = APP_DEMO_STATE_ALARM_SEND;
9    uint16_t slot = 0;
10   bool found_inputslot = false;
11   uint16_t subslot_ix = 0;
12   const app_subslot_t * p_subslot = NULL;
13   pnet_pnio_status_t pnio_status = {0};
14   pnet_diag_source_t diag_source = {
15      .api = APP_API,
16      .slot = 0,
17      .subslot = 0,
18      .ch = APP_DIAG_CHANNEL_NUMBER,
19      .ch_grouping = PNET_DIAG_CH_INDIVIDUAL_CHANNEL,
20      .ch_direction = APP_DIAG_CHANNEL_DIRECTION};
21
22   /* Loop though slots and subslots to find first input subslot */
23   while (!found_inputslot && (slot < PNET_MAX_SLOTS))
24   {
25      for (subslot_ix = 0; !found_inputslot && (subslot_ix <
       PNET_MAX_SUBSLOTS);
26           subslot_ix++)
27      {
28         p_subslot = &p_appdata->main_api.slots[slot].subslots[subslot_ix];
29         if (app_subslot_is_input (p_subslot))
30         {
31            found_inputslot = true;
32            break;
33         }
34      }
35      if (!found_inputslot)
36      {
37         slot++;
```

```
38      }
39   }
40   if (!found_inputslot)
41   {
42      printf ("Did not find any input module in the slots. Skipping.\n");
43      return;
44   }
45
46   diag_source.slot = slot;
47   diag_source.subslot = p_subslot->subslot_nbr;
48
49   switch (state)
50   {
51   case APP_DEMO_STATE_ALARM_SEND:
52      if (*p_alarm_allowed == true && arep != UINT32_MAX)
53      {
54         alarm_payload[0]++;
55         printf (
56            "Sending process alarm from slot %u subslot %u USI %u to "
57            "IO-controller. Payload: 0x%x\n",
58            slot,
59            p_subslot->subslot_nbr,
60            APP_ALARM_USI,
61            alarm_payload[0]);
62         pnet_alarm_send_process_alarm (
63            net,
64            arep,
65            APP_API,
66            slot,
67            p_subslot->subslot_nbr,
68            APP_ALARM_USI,
69            APP_ALARM_PAYLOAD_SIZE,
70            alarm_payload);
71         *p_alarm_allowed = false; /* Not allowed until ACK received */
72      }
73      else
74      {
75         printf ("Could not send process alarm, as alarm_allowed == false or
      "
76                 "no connection available\n");
```

```
77      }
78      break;
79
```

Boucle main :

la boucle se présentera comme telle :

```

if (flags & APP_EVENT_READY_FOR_DATA)
    {
        os_event_clr (p_appdata->main_events, APP_EVENT_READY_FOR_DATA);

        app_handle_send_application_ready (
            net,
            p_appdata->arep_for_appl_ready,
            p_appdata->arguments.verbosity);
```

```

  else if (flags & APP_EVENT_ALARM)
    {
        os_event_clr (p_appdata->main_events, APP_EVENT_ALARM); /* Re-arm
    */

        app_handle_send_alarm_ack (
            net,
            p_appdata->main_api.arep,
            p_appdata->arguments.verbosity,
            &p_appdata->alarm_arg);
    }
```

```

else if (flags & APP_EVENT_TIMER)
    {
        os_event_clr (p_appdata->main_events, APP_EVENT_TIMER); /* Re-arm
    */
        if (
            (p_appdata->main_api.arep != UINT32_MAX) &&
            (tick_ctr_update_data > APP_TICKS_UPDATE_DATA))
        {
            tick_ctr_update_data = 0;

            app_handle_cyclic_data (
                net,
```

```
            p_appdata,
            button1_pressed,
            data_ctr++,
            p_appdata->inputdata,
            sizeof (p_appdata->inputdata));


        }
    }
```

```
else if (flags & APP_EVENT_TIMER)
    {
        os_event_clr (p_appdata->main_events, APP_EVENT_TIMER); /* Re-arm
   */
        if (
           (p_appdata->main_api.arep != UINT32_MAX) &&
           (tick_ctr_update_data > APP_TICKS_UPDATE_DATA))
        {
           tick_ctr_update_data = 0;

           app_handle_cyclic_data (
              net,
              p_appdata,
              button1_pressed,
              data_ctr++,
              p_appdata->inputdata,
              sizeof (p_appdata->inputdata));

          ...
    }
```

.2 Annexe B : Script python pour capteur

Code pour la lecture du capteur DHT11 :

```
import Adafruit_DHT
import time
sensor = Adafruit_DHT.DHT11
DHT11_pin = 23

# Function for converting decimal to binary
def float_bin(my_number, places = 3):
    my_whole, my_dec = str(my_number).split(".")
    my_whole = int(my_whole)
    res = (str(bin(my_whole))+".").replace('0b','')

    for x in range(places):
        my_dec = str('0.')+str(my_dec)
        temp = '%1.20f' %(float(my_dec)*2)
        my_whole, my_dec = temp.split(".")
        res += my_whole
    return res

def IEEE754(n) :
    # identifying whether the number
    # is positive or negative
    sign = 0
    if n < 0 :
        sign = 1
        n = n * (-1)
    p = 30
    # convert float to binary
    dec = float_bin (n, places = p)

    dotPlace = dec.find('.')
    onePlace = dec.find('1')
    # finding the mantissa
    if onePlace > dotPlace:
        dec = dec.replace(".","")
        onePlace -= 1
        dotPlace -= 1
    elif onePlace < dotPlace:
```

```
38          dec = dec.replace(".","")
39          dotPlace -= 1
40      mantissa = dec[onePlace+1:]
41
42      # calculating the exponent(E)
43      exponent = dotPlace - onePlace
44      exponent_bits = exponent + 127
45
46      # converting the exponent from
47      # decimal to binary
48      exponent_bits = bin(exponent_bits).replace("0b",'')
49
50      mantissa = mantissa[0:23]
51
52      # the IEEE754 notation in binary
53      final = str(sign) + exponent_bits.zfill(8) + mantissa
54
55      # convert the binary to hexadecimal
56      hstr = '%0*X' %((len(final) + 3) // 4, int(final, 2))
57      a = '0x'+ hstr[0:2]
58      b = '0x'+ hstr[2:4]
59      c = '0x'+ hstr[4:6]
60      d = '0x'+ hstr[6:8]
61
62      return (a,b,c,d)
63
64  while True:
65
66      humidity, temperature = Adafruit_DHT.read_retry(sensor, DHT11_pin)
67      if humidity is not None and temperature is not None:
68          print('Temperature={0:0.1f}*C Humidity={1:0.1f}%'.format(
    temperature, humidity))
69          with open('sensor.txt',"w") as myfile:
70              for x in range(4):
71                  myfile.write(str(IEEE754(temperature)[x]) + '\n')
72
73      else:
74          ('Failed to get reading from the sensor. Try again!')
75      time.sleep(1)
```

.3 Annexe C : GSD file

- GSD file de notre IO device :

```
<VirtualSubmoduleList>
    <VirtualSubmoduleItem ID="13" SubmoduleIdentNumber="0x0001"
   MayIssueProcessAlarm="true">

     <IOData>
        <Input Consistency="All items consistency">
            <DataItem DataType="Float32" TextId="p_value" UseAsBits="false"
     />
        </Input>
     </IOData>

    <RecordDataList>
        <ParameterRecordDataItem Index="123" Length="4">
            <Name TextId="TOK_sample_parameter_1"/>
             <Ref DataType="Float32" ByteOffset="0" DefaultValue="1"
   AllowedValues="0..99" Changeable="true" Visible="true" TextId="Demo_1"/>
             </ParameterRecordDataItem>
             <ParameterRecordDataItem Index="124" Length="4">
            <Name TextId="TOK_sample_parameter_2"/>
             <Ref DataType="Float32" ByteOffset="0" DefaultValue="2"
   AllowedValues="0..999" Changeable="true" Visible="true" TextId="Demo_2
   "/>
              </ParameterRecordDataItem>
             </RecordDataList>
            <ModuleInfo>
             <Name TextId="TOK_Name_Module_I8O8"/>
             <InfoText TextId="TOK_InfoText_Module_I8O8"/>
              </ModuleInfo>
       </VirtualSubmoduleItem>
</VirtualSubmoduleList>
```

.4 Annexe D : Script linux pour test de performance

voici comment installer les dépendances nécessaires pour effectuer les tests :

```
sudo apt-get install -y build-essential
git clone git://git.kernel.org/pub/scm/utils/rt-tests/rt-tests.git
cd rt-tests
git checkout stable/v1.0
make all -j4
sudo make install
```

puis on utilise le script mklatencyplot

```
#!/bin/bash

# 1. Run cyclictest
cyclictest -l1000000 -m -Sp90 -i200 -h400 -q >output

# 2. Get maximum latency
max=`grep "Max Latencies" output | tr " " "\n" | sort -n | tail -1 | sed s
    /^0*//`

# 3. Grep data lines, remove empty lines and create a common field
    separator
grep -v -e "^#" -e "^$" output | tr " " "\t" >histogram

# 4. Set the number of cores, for example
cores=4

# 5. Create two-column data sets with latency classes and frequency values
    for each core, for example
for i in `seq 1 $cores`
do
  column=`expr $i + 1`
  cut -f1,$column histogram >histogram$i
done

# 6. Create plot command header
echo -n -e "set title \"Latency plot\"\n\
set terminal svg\n\
set xlabel \"Latency (us), max $max us\"\n\
set logscale y\n\
set xrange [0:400]\n\
```

```
28 set yrange [0.8:*]\n\
29 set ylabel \"Number of latency samples\"\n\
30 set output \"plot.svg\"\n\
31 plot " >plotcmd
32
33 # 7. Append plot command data references
34 for i in `seq 1 $cores`
35 do
36   if test $i != 1
37   then
38     echo -n ", " >>plotcmd
39   fi
40   cpuno=`expr $i - 1`
41   if test $cpuno -lt 10
42   then
43     title=" CPU$cpuno"
44    else
45     title="CPU$cpuno"
46   fi
47   echo -n "\"histogram$i\" using 1:2 title \"$title\" with histeps" >>
       plotcmd
48 done
49
50 # 8. Execute plot command
51 gnuplot -persist <plotcmd>
```

[17]

.5 Annexe E : IEEE754

Afin de préserver le maximum de bits de précision pour une taille de stockage donnée k, un nombre à virgule flottante v(ou à point flottant dans la version anglo-saxonne) est habituellement représenté sous la forme normalisée :

$$v = (+|-)1.x_{(k-1)}x_{(k-2)}...x_0 * 2^E$$

avec les x des chiffres binaires.

Le chiffre avant la virgule n'est jamais stocké et ne compte donc pas dans le calcul de la taille du format de représentation. La mise en forme normalisée peut nécessiter de déplacer le point flottant vers la gauche (lorsque est plus grand que 1 en valeur absolue) ou vers la droite (lorsque st plus petit que 1 en valeur absolue). [18]

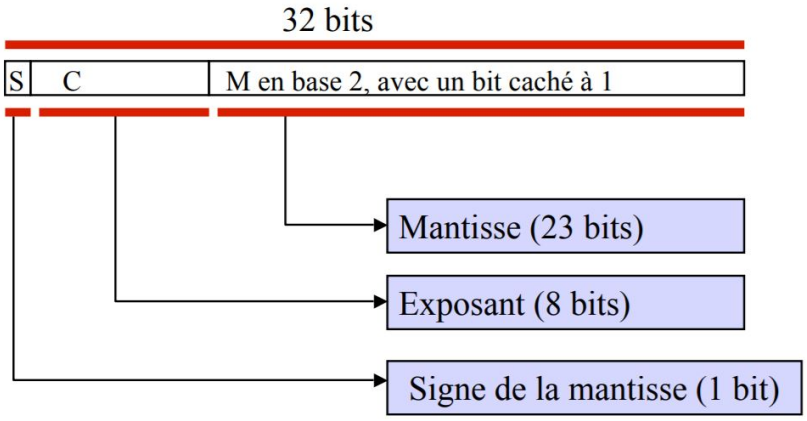

FIGURE 30 – Représentation en mémoire d'un nombre au format IEEE 754