Université de Lorraine

ÉCOLE DOCTORALE IAEM
Département de formation doctorale en informatique

Vérification des contraintes temporelles de bout-en-bout dans le contexte AutoSar

T H È S E

présentée et soutenue publiquement le 29 novembre 2012

pour l'obtention du

Doctorat de l'Université de Lorraine
(spécialité informatique)

par

Aurélien MONOT

Thèse dirigée par Françoise SIMONOT et Nicolas NAVET

préparée au LORIA, Projet TRIO

Jury :

Président :	Jean Charles FABRE	-	Professeur à l'INP-Toulouse
Rapporteurs :	Laurent GEORGE	-	Maître de conférence à l'Université de Paris Est
	Alain GIRAULT	-	Directeur de recherche à l'INRIA Rhône-Alpes
Autres membres :	Nicolas NAVET	-	Assistant Professeur à l'Université du Luxembourg
	Françoise SIMONOT	-	Professeur à l'Université de Lorraine
	Pierre-Étienne MOREAU	-	Professeur à l'Université de Lorraine
Invité :	Bernard BAVOUX	-	Expert en systèmes électroniques chez PSA Peugeot Citroën

Laboratoire Lorrain de Recherche en Informatique et ses applications – UMR7503

Remerciements

Cette thèse doit beaucoup aux nombreuses personnes qui m'ont encouragé, soutenu et conforté tout au long de ces années de thèse. Qu'elles trouvent dans ce travail l'expression de mes plus sincères remerciements.

En premier lieu, je souhaite remercier Françoise Simonot eu égard à son rôle de directrice de thèse, pour avoir toujours su se rendre disponible malgré un emploi du temps fort chargé et pour m'avoir prodigué de précieux conseils tout au long de ces travaux. Je lui suis très reconnaissant de la confiance qu'elle m'a accordée quand elle m'a proposé ce projet de thèse Cifre et pour son rôle à la fin de la thèse où elle a su trouver les mots pour m'encourager dans les moments difficiles de la rédaction.

Pour leur encadrement remarquable, j'aimerai remercier très chaleureusement mes deux autres co-encadrants de thèse, Nicolas Navet et Bernard Bavoux qui ont partagé ma garde respectivement à Nancy et à Vélizy tout au long des travaux de thèse. J'ai trouvé chez eux deux beaucoup d'enthousiasme et de gentillesse à mon égard ce qui a rendu nos discussions professionnelles et personnelles constructives et très agréables. Nicolas m'a appris beaucoup quant aux qualités d'un chercheur et Bernard a su me guider adroitement dans le monde de l'entreprise en me montrant les méthodes de travail dans un grand groupe et en me dirigeant vers les bons interlocuteurs parmi les nombreux collaborateurs chez PSA Peugeot Citroën.

Un grand merci également à Liliana Cucu-Grosjean, tour-à-tour mon enseignante, ma tutrice de stage, ma collègue de bureau, ma chef d'équipe et mon amie. Son dynamisme, son énergie et sa détermination dans la réalisation de ces projets m'ont poussé à aller de l'avant par émulation.

Je remercie fortement Marie-Françoise Loubressac, Laurence Bénini, Françoise Laurent et Véronique Bawedin qui ont été d'une efficacité et d'une patience modèles dans toutes les démarches liées à l'organisation de mes nombreux déplacements, ma navigation dans les méandres administratives et les petites choses de tous les jours.

Un grand merci aussi à Cristian Maxim pour avoir été particulièrement patient et à l'écoute de mes demandes lors des fastidieuses phases d'expérimentations avec le simulateur CAN et aussi à Rob Davis pour avoir pris le temps de m'écouter et de

me débloquer lors de mes études de simulation et d'analyse pour les réseaux CAN.

Je souhaite remercier mes collègues et amis au LORIA qui ont contribué à la bonne ambiance de l'équipe et du laboratoire : Olivier Bouré, Dorin Maxim, Cristian Maxim, Sylvain Raybaud, Pierre Caserta, Adrien Guénard, Code Lo, Christian Gillot, Bilel Nefzi, Fabrice Vergnaud, Hugo Cruz Sanchez, Luca Santinelli, Patrick Meumeu-Yomsi et Dominique Bertrand. Leur bonne humeur et les instants que nous avons partagés ensembles m'ont apporté beaucoup de joie.

De même, je souhaite remercier mes collègues de PSA Peugeot Citroën : Frédéric Mangonneaux, Vincent De Flaugergues, Matthieu Donain, Cécile Nocquet, Huy Cuong Nguyen, Ahmed Harrar et Didier Jampi. J'ai passé de bons moments en leur compagnie dans les bureaux de PSA Peugeot Citroën à Vélizy.

Merci beaucoup à Alain Girault et Laurent Georges qui m'ont fait l'honneur d'être rapporteurs de ma thèse ainsi qu'à Jean-Charles Fabre pour avoir accepté de présider le jury de thèse. Je remercie aussi chaleureusement Pierre-Etienne Moreau, mon référent interne au LORIA pour avoir accepté de faire parti du jury de thèse et d'apporter un oeil extérieur à l'informatique embarquée.

À Lucien Lecocq, Marie Lepoutre, Sylvain Le Net, Audrey Dot, Alexandre Hucher, Christophe Sigaud, Cyrille Chopelet, Maude Liotard, Charlotte Rosak, Diana Dos Santos, Etienne Friedli, Cédric Favre, Ksenia et Michael Wahler, Lucas Bonfort et ceux qui se reconnaîtront, merci pour les bon moments passés avant et pendant ma thèse, malgré les distances.

Enfin, un grand merci du fond du cœur à ma famille, en particulier à mes parents pour leur soutien infaillible à tous les points de vue pendant toute la durée de mes études culminant avec ce doctorat.

Résumé

Les systèmes électroniques embarqués dans les véhicules ont une complexité sans cesse croissante. Cependant, il est crucial d'en maîtriser le comportement temporel afin de garantir la sécurité ainsi que le confort des passagers. La vérification des contraintes temporelles de bout-en-bout est donc un enjeu majeur lors de la conception d'un véhicule. Dans le contexte de l'architecture logicielle AUTOSAR standard dans les véhicules, nous décomposons la vérification d'une contrainte de bout-en-bout en sous-problèmes d'ordonnancement sur les calculateurs et sur les réseaux de communication que nous traitons ensuite séparément.

Dans un premier temps, nous présentons une approche permettant d'améliorer l'utilisation des calculateurs exécutant un grand nombre de composants logiciels, compatible avec l'introduction progressive des plateformes multi-coeurs. Nous décrivons des algorithmes rapides et efficaces pour lisser la charge périodique sur les calculateurs multi-coeurs en adaptant puis en améliorant une approche existant pour les bus CAN. Nous donnons également des résultats théoriques sur l'efficacité des algorithmes dans certains cas particuliers. Enfin, nous décrivons les possibilités d'utilisation de ces algorithmes en fonction des autres tâches exécutées sur le calculateur.

La suite des travaux est consacrée à l'étude des distributions de temps de réponse des messages transmis sur les bus CAN. Dans un premier temps nous présentons une approche de simulation basée sur la modélisation des dérives d'horloges des calculateurs communicant sur le réseau. Nous montrons que nous obtenons des distributions de temps de réponse similaires en réalisant une longue simulation avec des dérives d'horloge ou en faisant un grand nombre de courtes simulations sans dérives d'horloge. Nous présentons enfin une technique analytique pour évaluer les distributions de temps de réponse des trames CAN. Nous présentons différents paramètres d'approximation permettant de réduire le nombre très important de calculs à effectuer en limitant la perte de précision. Enfin, nous comparons expérimentalement les résultats obtenus par analyse et simulation et décrivons les avantages et inconvénients respectifs de ces approches.

Mots-clés : systèmes temps-réel, électronique embarquée, AUTOSAR, ordonnancement, calculateurs multi-cœurs, réseaux CAN, offsets, lissage de charge, distribution de temps de réponse, dérive d'horloge

Abstract

The complexity of electronic embedded systems in cars is continuously growing. Hence, mastering the temporal behavior of such systems is paramount in order to ensure the safety and comfort of the passengers. As a consequence, the verification of end-to-end real-time constraints is a major challenge during the design phase of a car. The AUTOSAR software architecture drives us to address the verification of end-to-end real-time constraints as two independent scheduling problems respectively for electronic control units and communication buses.

First, we introduce an approach, which optimizes the utilization of controllers scheduling numerous software components that is compatible with the upcoming multicore architectures. We describe fast and efficient algorithms in order to balance the periodic load over time on multicore controllers by adapting and improving an existing approach used for the CAN networks. We provide theoretical result on the efficiency of the algorithms in some specific cases. Moreover, we describe how to use these algorithms in conjunction with other tasks scheduled on the controller.

The remaining part of this research work addresses the problem of obtaining the response time distributions of the messages sent on a CAN network. First, we present a simulation approach based on the modelisation of clock drifts on the communicating nodes connected on the CAN network. We show that we obtain similar results with a single simulation using our approach in comparison with the legacy approach consisting in numerous short simulation runs without clock drifts. Then, we present an analytical approach in order to compute the response time distributions of the CAN frames. We introduce several approximation parameters to cope with the very high computational complexity of this approach while limiting the loss of accuracy. Finally, we compare experimentally the simulation and analytical approaches in order to discuss the relative advantages of each of the two approaches.

Key-words: real-time embedded systems, AUTOSAR, scheduling, multicore controllers, CAN network, offsets, load balancing, response time distribution, clock drift

Table des matières

Table des matières

Chapitre 1

Introduction

Sommaire

1.1 Contexte de la thèse

1.1.1 Systèmes embarqués dans le domaine automobile

Depuis l'invention du premier véhicule par Joseph Cugnot en 1769, l'introduction de l'électronique dans les véhicules fait incontestablement partie des ruptures technologiques majeures dans l'histoire automobile. La première utilisation d'un système embarqué dans une voiture afin de commander le système d'injection de carburant en 1967 marque le début d'une nouvelle ère dans l'industrie automobile. Progressivement, de nouvelles fonctionnalités reposant sur l'électronique ont été intégrées dans les véhicules afin de remplacer, assister ou piloter des systèmes mécaniques ou hydrauliques. L'arrivée de composants électronique embarqués dans les voitures a permis progressivement de rendre les véhicules plus sûrs (ABS, ESP), plus efficaces (contrôle de l'injection, véhicules hybrides), et plus confortables (climatisation automatique, lève-vitre, régulateur de vitesse, GPS). La voiture, une invention thermique et mécanique, est devenue à présent un vrai ordinateur sur roues dont la puissance de calcul dépasse par plusieurs ordres de magnitude celle du système de guidage utilisé par la NASA lors de la mission Apollo sur la Lune.

Naturellement, l'intégration des systèmes embarqués dans les voitures s'est faite progressivement. Ainsi, aux débuts de l'électronique embarquée dans les véhicules, chaque calculateur électronique était dédié à une fonction et chaque calculateur était produit par une seule entité, que ce soit le constructeur automobile ou un fournisseur. Ensuite, avec l'émergence de nouvelles fonctions pilotées par électronique, certains calculateurs se sont vus en charge de l'exécution de plusieurs de ces fonctions afin d'éviter la prolifération de calculateurs dans les véhicules. Dans un premier temps, la conception entière de chaque calculateur était encore réalisée par le constructeur ou par un équipementier. Cependant, avec la multiplication de nouvelles fonctions reposant sur l'électronique embarquée, ce modèle a dû évoluer de manière à pouvoir intégrer au sein d'un même calculateur des fonctions réalisées par plusieurs partenaires distincts. Un grand nombre de ces fonctions requiert la mise-en-oeuvre de sous-modules répartis sur plusieurs calculateurs qui doivent alors communiquer entre eux. Afin de limiter le nombre de connexions filaires point à point entre les calculateurs et les capteurs, les automobiles intègrent des systèmes de communications multiplexés qui permettent à un ensemble de calculateurs de s'échanger des messages via un bus commun. Avec le temps, l'architecture électronique des voitures est ainsi devenue très complexe et nécessite la collaboration d'un grand nombre de partenaires différents afin de concevoir et réaliser le matériel et les composants logiciels nécessaires au bon fonctionnement de l'ensemble.

L'électronique embarquée est un enjeu majeur dans l'industrie automobile. De nos jours, une voiture sortant d'usine comporte plusieurs dizaines de calculateurs, appelés "Electronic Control Unit" (ECU), distribués dans tout le volume du véhicule et communicants entre eux via plusieurs bus de communication de type CAN

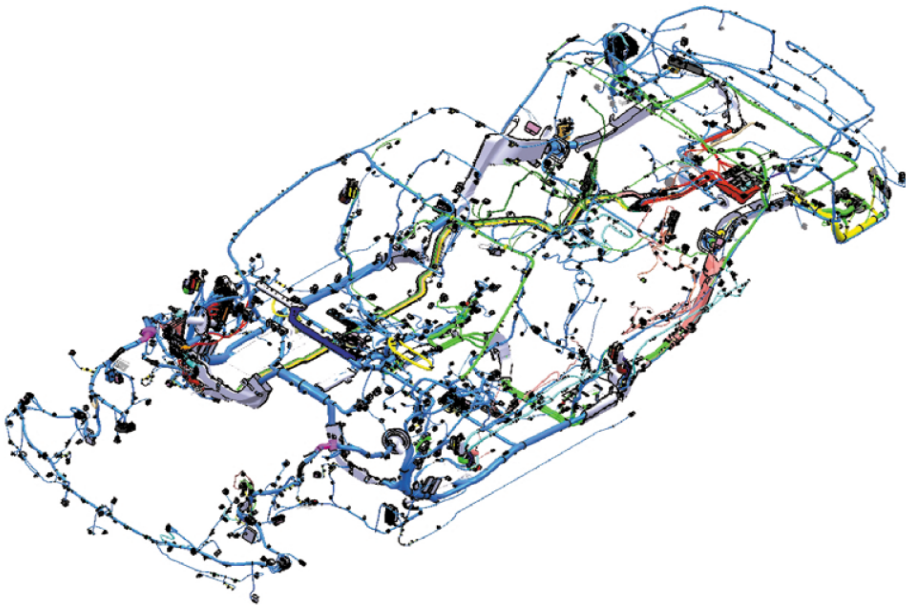

Figure 1.1 – Schéma du câblage électronique dans une BMW série 7 [25]

(Controller Area Network). D'autres protocoles de communication sont parfois utilisés dans les véhicules comme par exemple : LIN une alternative bas-coût à CAN, FlexRay dans certains véhicules haut de gamme ou MOST pour des applications multimédia. Cependant, les réseaux CAN sont prédominants dans les architectures électroniques des véhicules avec 750 millions de calculateurs capable de communiquer sur CAN vendus en 2010. La figure 1.1 montre le schéma d'implantation des faisceaux électroniques dans un véhicule haut de gamme. On estime que 40% du coût de production d'un véhicule est lié aux composants matériels électroniques et aux composants logiciels. De plus, 90% des innovations faites dans le domaine automobile reposent sur l'électronique embarquée. Dans cette mesure, les constructeurs automobiles mettent beaucoup d'effort à maîtriser la complexité de l'architecture électronique de leur voiture afin de garantir son bon fonctionnement ainsi qu'à l'optimiser et la simplifier afin de réduire les coûts de production.

1.1.2 Les grands challenges

Le nombre de composants électroniques embarqués dans les voitures de nos jours soulèvent ainsi quelques grands défis pour les constructeurs automobiles :

– **l'inter-opérabilité** entre les composants électroniques. Le problème se pose entre autre sur les plans logiciels et matériels ainsi que pour l'adéquation entre logiciel et matériel. Un des problèmes importants est l'inter-opérabilité entre

composants logiciels provenant de différentes sources mais c'est précisément pour cela qu'AUTOSAR (présenté en 1.1.3) est en train d'être déployé.

- **la sûreté fonctionnelle** des composants électroniques. Il s'agit d'un problème compliqué au regard de la complexité des systèmes utilisés. En particulier, il peut être difficile d'évaluer la sûreté de fonctionnement d'un calculateur ayant une architecture matérielle moderne et un nombre très importants de composants logiciels multi-sources. C'est pour répondre à ce défi que l'ISO 26262 a été définie.
- **l'optimisation**. Il s'agit d'un challenge qui intervient à différents niveaux d'échelle et à de nombreux composants d'un véhicule. L'enjeu principal derrière le souci d'optimisation est la réduction des coûts. Les architectures logicielles et matérielles doivent être dimensionnées au plus juste au moment de la conception car elles impactent des millions de véhicules pour les constructeurs généralistes. Ainsi, il est primordial de pouvoir faire plus avec le même matériel ou alors de pouvoir choisir une plateforme avec moins de mémoire ou un processeur moins cher lorsque les conditions le permettent.
- **la vérification des performances temporelles** dès les premières phases de conception des fonctions pilotées par les systèmes électroniques. Il est important de maîtriser les temps de réponse des fonctions liées à la sécurité du véhicule mais aussi pour les fonctions de confort pour lesquelles l'agrément de l'utilisateur peut être une source d'avantage concurrentiel. Cela implique d'étudier de nombreux sous-problèmes dans différents domaines que ce soit la traçabilité des exigences temporelles lors de la conception, la prédiction et l'évaluation temporelle des différents composants ou le calcul de temps de réponse dans le cas de ressources partagées.

Ce dernier challenge est transverse aux trois premiers. En effet, l'inter-opérabilité se pose aussi au niveau temporel lorsque plusieurs composants sont associés ; certains aspects de sûreté fonctionnelle peuvent reposer sur les performances temporelles de certains composants ; l'optimisation et le dimensionnement des composants électroniques nécessitent aussi de pouvoir valider les performances temporelles des systèmes étudiés. Les travaux de cette thèse ont donc pour sujet cette dernière problématique.

1.1.3 Normes et standards

Avec l'importance croissante des systèmes électroniques dans les voitures, diverses initiatives de normalisation et standardisation ont vu le jour. Dans le contexte de cette thèse, il convient en particulier de mentionner la norme ISO26262 [3], le consortium AUTOSAR [1] (AUTomotive Open System ARchitecture) et les projets TIMMO/TIMMO-2-USE. La première est une norme de sûreté de fonctionnement spécifique aux systèmes embarqués dans les voitures. Le second est un partena-

riat entre constructeurs automobiles, équipementiers et autres entreprises spécialisées dans ce domaine visant à établir un standard pour l'architecture logicielle des calculateurs automobiles et leur développement. Enfin, le projet TIMMO et sa seconde itération TIMMO-2-USE [2] abordent des aspects temporels non traités dans AUTOSAR. Les résultats de ces projets seront en parties réutilisées dans les mises-à-jour du standard AUTOSAR.

ISO 26262

L'ISO 26262 est l'application spécifique aux véhicules routiers de la norme générique CEI 61508 qui traite de la sécurité fonctionnelle des systèmes électriques et électroniques programmables. Elle définit des niveaux d'intégrité pour les composants des systèmes embarqués dans les véhicules qui prennent en compte, dans l'analyse des risques, aussi bien les aspects quantitatifs que qualitatifs. Ces niveaux d'intégrité, appelés ASIL (Automotive System Integrity Level) sont au nombre de quatre : ASIL A à D, respectivement du plus bas au plus haut niveau de criticité. Il existe également un niveau QM (Quality Management) dans le cas d'un système non influent sur la sécurité. Ces niveaux tiennent en compte la probabilité d'un risque, sa sévérité (gravité des blessures, et sa contrôlabilité par un conducteur moyen. Ces différents niveaux sont associés à différentes exigences de vérification des composants afin d'être certifiés. L'objectif de la norme est de standardiser les pratiques de test et de vérification des composants électroniques dans l'industrie automobile. Le tableau 1.1 donne les niveaux ASIL suivant les valeurs des différents paramètres.

Sévérité	Fréquence	Contrôlabilité (par un conducteur moyen)		
	par an d'utilisation	Simple (>99%)	Normale (>90%)	Incontrôlable
Pas de blessés	f < 0,1%	QM	QM	QM
	0,1% < f < 1%	QM	QM	QM
	1% < f < 10 %	QM	QM	ASIL A
	10 % < f	QM	ASIL A	ASIL B
Blessés légers	< 0,1%	QM	QM	QM
	0,1% < f < 1%	QM	QM	ASIL A
	1% < f < 10 %	QM	ASIL A	ASIL B
	10 % < f	ASIL A	ASIL B	ASIL C
Blessés graves	< 0,1%	QM	QM	ASIL A
	0,1% < f < 1%	QM	ASIL A	ASIL B
	1% < f < 10 %	ASIL A	ASIL B	ASIL C
	10 % < f	ASIL B	ASIL C	ASIL D

Table 1.1 – Niveaux d'intégrité ASIL

La mise en oeuvre imminente de cette norme entraîne certains choix de conception comme l'isolation de composants logiciels provenant de différentes sources comme nous le verrons dans le chapitre 2.

AUTOSAR

AUTOSAR [1] est né, entre autres raisons, de la volonté des différents acteurs dans le domaine de l'électronique automobile de faciliter le développement de systèmes intégrant des composants provenant de partenaires différents et de permettre la réutilisation de composants logiciels. Cet effort a donné lieu à la définition d'architectures, de briques de base logicielles et d'interfaces standardisées. Cela permet de garantir l'inter-opérabilité des composants logiciels répondant aux spécifications du standard tout en gardant un aspect concurrentiel sur l'implémentation des différentes briques logicielles. Depuis le début du projet en 2003 jusqu'à présent, le standard a connu plusieurs révisions dont la version le plus récente est la 4.0.3 publiée en janvier 2012.

Le standard repose sur une approche d'ingénierie logicielle à base de composants et spécifie une méthodologie à suivre pour concevoir un calculateur conforme au standard. L'architecture logicielle, présentée en figure 1.2 se décompose en trois grandes couches permettant de découpler les aspects fonctionnel et matériel :

- la couche "Basic Software" qui contient l'OS (AUTOSAR OS héritier de OSEK/VDX [60]) et divers services de bas-niveau comme la communication. Certains des composants logiciels de cette couche ("Basic Software Components" ou BSW Components), comme par exemple les drivers, sont dépendants de la plateforme matérielle sur laquelle ils sont exécutés. Cette couche doit donc être développée en accord avec la plateforme d'exécution mais permet précisément aux couches supérieures d'être indépendantes du matériel.
- la couche "Application Software" qui contient le logiciel spécifique aux composants fonctionnels ("Application Software Components" ou ASW Components). C'est ici que se situe le code mettant en oeuvre les fonctions exécutés sur la plateforme. Cette couche est indépendante de la plateforme matérielle sous-jacente.
- la couche "Runtime Environment" (RTE) qui sert à faire communiquer les composants applicatifs (ASW) entre eux ou avec les composants de la couche inférieure (BSW)

La couche RTE sert de tampon entre la couche applicative et indépendante de la plateforme et les services de bas niveau, spécifiques à la plateforme. De ce fait, l'exécution des composants logiciels et applicatifs et l'envoi des messages contenant les données entrantes et sortantes des fonctions distribuées sur un réseau sont indépendants et asynchrones. Cela permet de séparer les problèmes et donc faciliter la conception. En revanche, il n'est pas possible de faire d'hypothèse précise

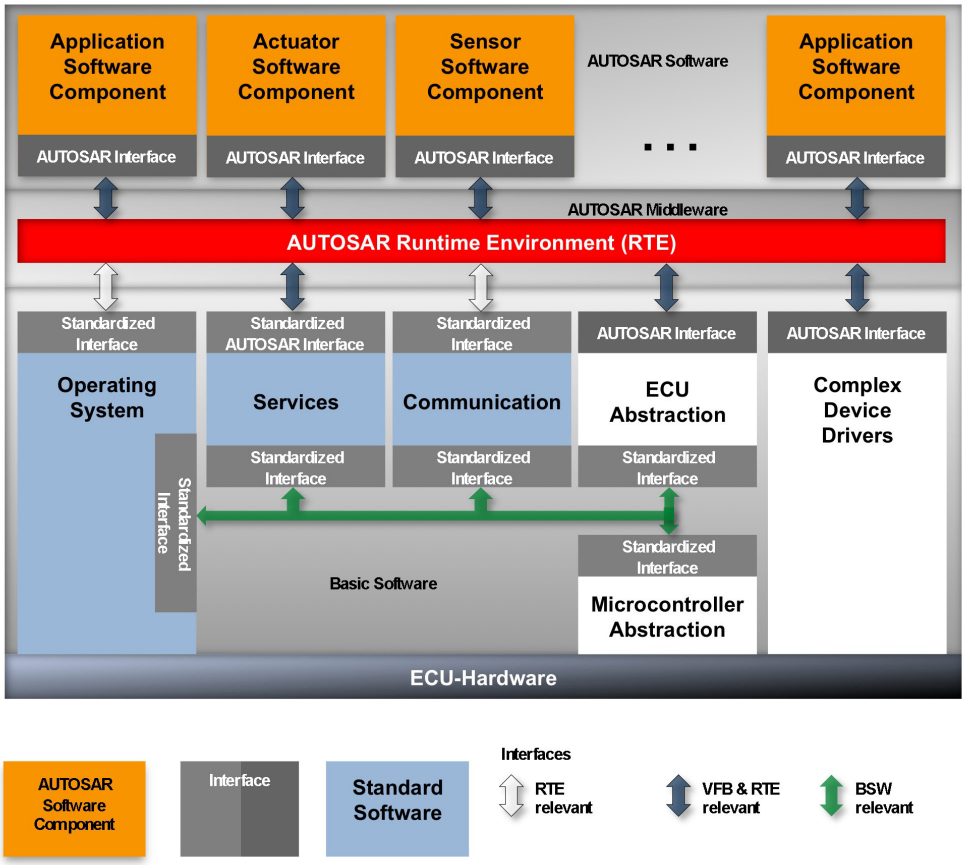

Figure 1.2 – Architecture logicielle d'un calculateur AUTOSAR. Figure extraite de [1]

sur les délais d'attente pour qu'une information traverse cette couche si ce n'est une hypothèse pire cas pessimiste. Ceci motivera à séparer les travaux de thèse en deux parties : une première s'intéressant aux composants logiciels dans les calculateurs et une seconde s'intéressant aux messages communiqués sur les réseaux.

TIMMO et TIMMO-2-USE

Bien qu'AUTOSAR soit utilisé depuis plusieurs années, le standard actuel ne couvre toujours pas l'ensemble des informations et des besoins nécessaires au processus de développement complet d'une automobile. En particulier, la maîtrise des aspects temporels dans le processus de développement sont devenues cruciales pour la conception de systèmes automobiles distribués. Contrairement aux aspects purement fonctionnels et à la fiabilité, les contraintes temporelles sont généralement considérées tard dans le cycle de développement (typiquement lors de l'intégration) et ce, sans qu'il existe une approche standardisée. De plus, le comportement temporel est généralement appréhendé plus sous l'angle de la mesure et du test que des approches formelles et de l'analyse. En conséquence, les fonctions innovantes ne sont pas implémentées, ou ne sont pas implémentées de façon efficace du point de vue de l'optimisation des coûts et des temps de développement.

Plusieurs initiatives ont été lancées pour capturer et modéliser une variété de facettes des architectures électroniques automobiles, on peut citer DECOS, EASIS, StepX, GraForSys, GeneralStore, EAST-ADL2. Néanmoins, le temps n'a jamais été considéré ou il l'a été de façon simplifiée sans considérer les contraintes d'intégration chez les constructeurs de collaboration entre les différents acteurs ou les contraintes liées à la distribution des applications sur les réseaux.

Le standard AUTOSAR est certainement un pas en avant pour gérer la complexité des systèmes automobiles et il a été une première fois étendu dans la dimension temporelle dans sa dernière version (V4.0). Ces extensions ont été élaborées en coopération étroite avec le projet TIMMO qui visait à modéliser et analyser les aspects temporeles dans l'électronique embarquée dans les véhicules.. Suite à ce premier projet, une seconde itération sur ces travaux a été lancée sous la forme du projet TIMMO-2-USE [2] qui vise à améliorer et optimiser les résultats du premier projet en travaillant entre autre à rendre l'approche plus facilement utilisable comme par exemple en renforçant les aspects d'automatisation dans la méthodologie issue de ces travaux.

Les travaux de cette thèse s'inscrivent dans ces efforts pour maîtriser les contraintes temporelles dans le processus de développement des systèmes électroniques embarqués dans le véhicule. Ainsi les approches et algorithmes résultants de cette thèse sont utilisables afin de répondre à certains cas d'usage de TIMMO-2-USE qui couvre un domaine beaucoup plus large. En effet, les aspects méthodologiques, comme par exemple la traçabilité des exigences temporelles d'un niveau de modélisation à un

autre ou la négociation de budget temporels entre les différents intervenants, ne sont pas spécifiquement traités dans cette thèse car cela a déjà été fait dans les projets TIMMO et TIMMO-2-USE.

1.2 Une approche système du véhicule

Dans le contexte de cette thèse, nous nous intéressons aux "fonctions pilotées", c'est à dire aux fonctions mises en oeuvre par des composants électroniques embarqués dans la voitures (capteurs, actionneurs, calculateurs et réseaux). Nous étudions en particulier les problématiques de contraintes de temps de bout-en-bout associées à ces fonctions pilotées. Après avoir présenté le concept de contrainte de temps de bout-en-bout, nous détaillons comment cette contrainte se décompose dans le contexte du processus de conception des fonctions pilotées pour finalement s'intéresser à l'ordonnancement des composants logiciels sur les calculateurs et des messages sur les réseaux de communication.

1.2.1 Contrainte de temps de bout-en-bout

La conception d'une fonction pilotée chez un constructeur automobile suit une méthodologie classique de type cycle en V. Ainsi, une des premières étapes consiste à spécifier les besoins. Lors de cette étape, les concepteurs rédigent un document listant un certain nombre d'exigences à réaliser dans différents domaines parmi lesquelles on trouve généralement

- des exigences fonctionnelles, qui expriment ce que réalise la fonction,
- des exigences d'intégration, qui assurent la compatibilité avec l'existant,
- des exigences de sûreté de fonctionnement, qui expriment les besoins de fiabilité, de sécurité et des autres aspects de ce domaine,
- des exigences temporelles, qui expriment les besoins en performances temporelles des fonctions.

Dans le cadre des travaux de thèse, c'est à ces dernières que nous nous intéressons sous la forme de "contrainte de temps de bout-en-bout". Ici, le terme "de bout-en-bout" exprime que nous nous intéressons à ce qu'il se passe entre le stimulus, qui initie la fonction étudiée, et la réaction, à la fin de l'exécution de la fonction. Ainsi, dans le contexte de l'utilisation d'une voiture, nous étudions ce qu'il se passe temporellement dans l'architecture électronique embarquée dans le véhicule depuis un capteur jusqu'à un actionneur. Ces exigences portent généralement sur des délais de réponse entre le stimulus et la réaction et peuvent se modéliser comme représenté sur la figure 1.3.

Les contraintes de temps de bout-en-bout sont utilisées par le constructeur pour traduire les besoins de performances qui servent en particulier à deux aspects très importants pour les clients : la sécurité et l'agrément des utilisateurs. Pour une

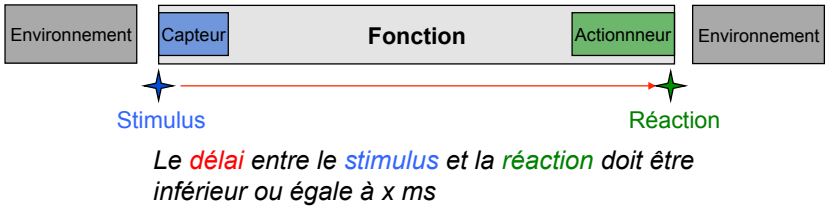

Figure 1.3 – Contrainte de temps de bout-en-bout. Le point de départ est situé à l'interface environnement/capteur et le point d'arrivée à l'interface actionneur/environnement.

même fonction sera alors détaillée une liste d'exigences de performances. Voici deux exemples de contraintes de bout-en-bout liées traduisant une exigence de sécurité pour la fonction responsable du freinage du véhicule :

- *"Le délai maximum entre une pression sur la pédale de frein et l'allumage des feux stop doit être inférieur ou égal à 50 ms"*
- *"Le délai maximum entre une pression sur la pédale de frein et l'instant où les plaquettes de frein entrent en contact avec le disque du frein doit être inférieur ou égal à 20 ms"*

Les fonctions pilotées sont généralement distribuées, c'est à dire qu'elles mettent en oeuvre plusieurs capteurs, calculateurs et actionneurs répartis dans le véhicule et communiquant entre eux. Comme montré dans l'exemple précédent, les différentes contraintes de temps de bout-en-bout d'une même fonction peuvent être associées à des stimuli et réactions différentes. Ainsi, les flux de donnée correspondant à ces différentes contraintes vont suivre des chemins différents. Dans le contexte de l'étude des contraintes de temps, ces chemins sont appelés "chaînes temporelles". Lors de l'étude de ces contraintes, il faut donc s'intéresser à ces chaînes une par une.

1.2.2 Décomposition au niveau fonctionnel

Lors de l'étape suivante du développement d'une fonction pilotée, les concepteurs procèdent au raffinement de la fonction pilotée étudiée en la décomposant en sous-fonctions. Ainsi lors de la phase de conception d'architecture fonctionnelle, la fonction est décomposée jusqu'à obtenir des blocs fonctionnels atomiques. Cette décomposition est finie lorsque les blocs fonctionnels correspondent au niveau de granularité des composants logiciels élémentaires. A ce deuxième niveau de description, il est alors possible de décomposer une contrainte de bout-en-bout en plusieurs sous-contraintes associées à chacun des éléments de la chaîne temporelle. La figure 1.4 montre un exemple de chaîne temporelle une fois la décomposition de l'architecture fonctionnelle réalisée.

Dans cet exemple, pour une contrainte sur un délai de bout-en-bout de valeur T, on a la décomposition suivante en sous-contraintes auxquelles on associe les délais intermédiaires de valeurs suivantes : c pour le capteur, x_i pour les blocs fonctionnels atomiques, y_i pour communiquer entre ces blocs et a pour l'actionneur. La contrainte sera respectée si la somme de ces valeurs est inférieure à T.

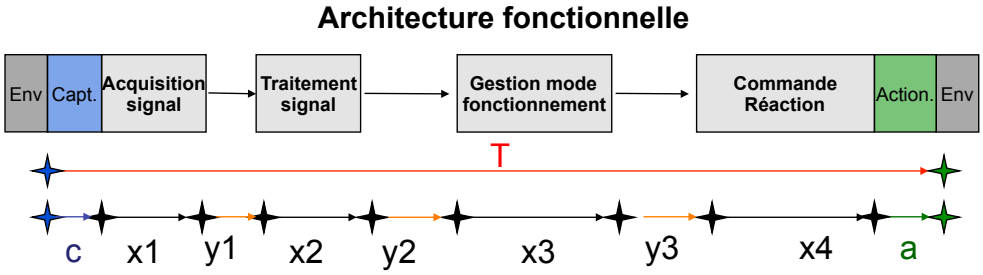

Figure 1.4 – Exemple de chaîne temporelle associée à une décomposition fonctionnelle. La chaîne temporelle est décomposée en sous-segment pour chacun des blocs fonctionnels. Pour assurer la cohérence, il faut que $c + x1 + y1 + x2 + y2 + x3 + y3 + x4 + a \leq T$

Dans le cas général, les blocs fonctionnels ont plusieurs signaux (données) en entrée et peuvent avoir également plusieurs signaux en sortie. En conséquence, la décomposition strictement fidèle d'une contrainte de bout-en-bout en contraintes intermédiaires n'est pas nécessairement linéaire comme dans le cas de notre exemple. Dans la pratique, les concepteurs de la fonction étudiée savent identifier un chemin critique lors de la décomposition au niveau fonctionnel. Cette connaissance est nécessaire et utilisée afin d'obtenir une décomposition en contraintes intermédiaires linéaires.

1.2.3 Architecture opérationnelle

De manière courante, la conception de l'architecture matérielle, c'est à dire la liste des calculateurs et leur topologie est réalisée en parallèle du développement de chacune des fonctions pilotées. Ainsi, une fois la phase de conception de l'architecture fonctionnelle réalisée, les concepteurs procèdent à la réalisation de l'architecture opérationnelle qui correspond à l'allocation des blocs fonctionnels sur l'architecture matérielle. Ceci est possible car le niveau de granularité de l'architecture fonctionnelle assure que les sous-fonctions correspondent à des blocs logiciels. En revanche, cela peut nécessiter la définition de nouveaux messages échangés sur les réseaux pour transférer les informations entre les fonctions réparties sur des calculateurs différents.

La figure 1.5 montre la décomposition de la chaîne temporelle étudiée au niveau

de l'architecture opérationnelle. Les différents blocs fonctionnels sont distribués sur trois calculateurs distincts communicant via un bus CAN. De plus, les deux premières sous-fonction de la chaîne sont situés sur le même calculateur. Suivant le fonctionnement des capteurs et des actionneurs, différentes décompositions peuvent être effectuées en ajoutant ou non un délai entre ceux-ci et les sous-blocs fonctionnels qui leur sont connectés. Ici, on considère que le capteur fonctionne de manière synchrone avec le bloc fonctionnel "acquisition signal". En conséquence la décomposition reste la même.

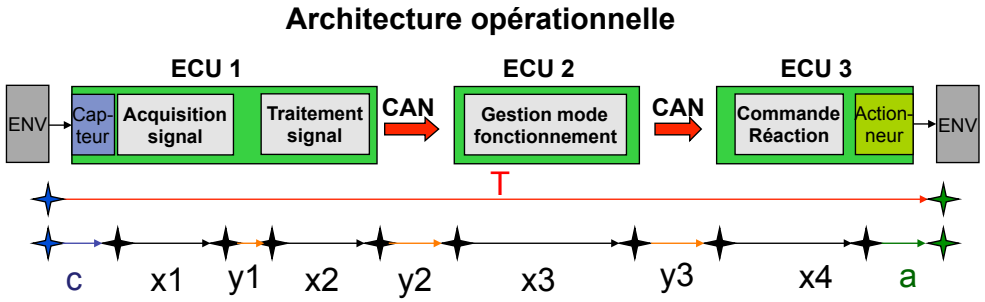

Figure 1.5 – Description de l'exemple de chaîne temporelle étudié au niveau opérationnel lequel fait apparaître l'allocation des composants fonctionnels sur l'architecture matérielle.

1.2.4 Transformation en architecture logicielle AUTOSAR

Une fois l'architecture opérationnelle décrite, la transformation pour avoir l'architecture logicielle AUTOSAR est directe. En effet, chaque bloc fonctionnel correspond à un composant logiciel atomique de type "Atomic Software Component" (A-SWC). Ces A-SWC sont mis en oeuvre sous la forme de "runnables" qui sont des composants logiciels élémentaires ayant un flot d'exécution séquentiel. Un A-SWC peut être implémenté par un ensemble de plusieurs runnables mais pour la clareté de l'exemple, nous faisons l'hypothèse que chaque bloc fontionnel ne nécessite qu'un runnable. Si on considère que les runnables F1, F2, F3 et F4 sont les runnables qui mettent en oeuvre les quatre sous-fonctions, la flèche verte en pointillés dans la figure 1.6 montre alors le chemin suivi par le signal dans l'architecture AUTOSAR correspondant à notre exemple.

Sur la base de cette architecture, il est alors possible de décomposer encore une fois la chaîne temporelle en segmentant le chemin parcouru par le signal couche par couche comme illustré dans la figure 1.7. Chaque segment correspond au délai entre l'instant ou l'information est transmise par la couche précédente à l'instant ou le signal est transmis à la couche suivante. La vérification de la contrainte de

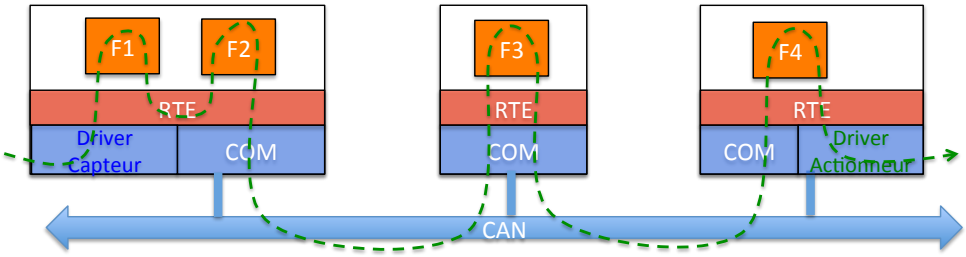

Figure 1.6 – Description de l'exemple de chaîne temporelle étudié sur une architecture logicielle de type AUTOSAR. La flèche pointillée en vert montre le trajet du signal du capteur à l'actionneur à travers l'architecture logicielle.

bout-en-bout nécessite d'étudier les délais sur chacun des segments. Néanmoins, il est possible de faire quelques simplifications pour certains de ces segments en s'appuyant sur le fonctionnement d'un calculateur. Ainsi, les échanges de données entre composants logiciels situés sur le même calculateur se font généralement sous la forme de variables écrites sur une mémoire partagée dont les accès en lecture et écriture sont fait de manière synchrone lors de l'exécution des runnables correspondants. Dans notre exemple, on peut donc négliger le passage par le RTE entre F1 et F2 c'est à dire l'ancien segment y1. Pour les mêmes raisons, les accès au capteur et à l'actionneur via le RTE sont faits respectivement pendant l'exécution de F1 et F4. Les délais de fonctionnement du capteur et de l'actionneur sont gardés de manière explicite mais pourraient être intégrés aux temps T1 et T4.

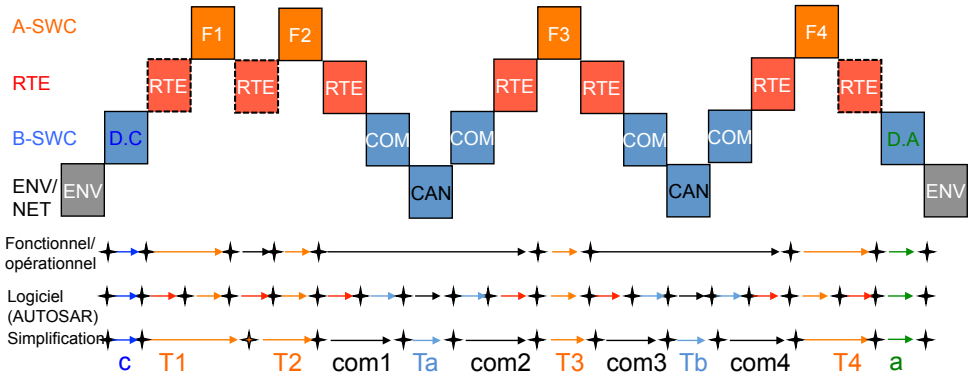

Figure 1.7 – Décomposition de la chaîne temporelle étudiée en fonction du parcours à travers les couches de l'architecture AUTOSAR. Les blocs représentés dans cette figure correspondant aux intervalles de temps passés dans chacune des couches et incorporent les éventuels temps d'attente (asynchronismes, ordonnancement...)

Nous avons donc quatre types de segments une fois arrivé à ce niveau de description :

- les délais de fonctionnement des organes physiques (capteurs et actionneurs), ici *c* et *a.*
- les temps de réponse des composants logiciels (les runnables), ici *T1*, *T2*, *T3* et *T4.*
- les temps de transition du signal entre la couche applicative et la couche de communication, ici *com1*, *com2*, *com3* et *com4.*
- le temps de réponse des messages émis sur les réseaux, ici *Ta* et *Tb.*

En ce qui concerne le premier type de segment, les délais de fonctionnement des actionneurs et des capteurs ne seront pas étudiés ici. Il est supposé que ces informations sont obtenues par mesure ou par le fournisseur du composant. Les autres types de segment par contre nécessitent de regarder de plus près ce qu'il se passe au niveau de l'exécution de runnable, ainsi qu'au niveau de la transmission de messages sur les réseaux et enfin aux transitions entre la couche logicielle applicative et la couche responsable de la communication situées de part et d'autre du RTE.

1.2.5 Partage de ressources et ordonnancement

A présent, prenons l'exemple de deux chaînes temporelles correspondant à deux contraintes temporelles de bout-en-bout *F* et *G*. Ces deux contraintes pourraient être associées à des fonctions pilotées différentes ou à deux exigences de performance pour une seule fonction pilotée. Ces deux contraintes se décomposent chacune en trois sous-fonctions, respectivement *F1*, *F2*, *F3* et *G1*, *G2*, G3 dont les runnables correspondant sont distribués comme indiqué dans la figure 1.8. *F1*, *F2* et *G1* partagent un même calculateur, ainsi que *F3* et *G3*. De plus les messages de *F2* à *F3*, de *G1* à *G2*, et de *G2* à *G3* sont émis sur le même bus CAN. Comme il n'est possible de n'exécuter qu'une seul tâche à la fois sur un processeur et de ne transmettre qu'un seul message à la fois sur un bus, il devient alors nécessaire d'ordonnancer les runnables sur les calculateurs et les messages sur les réseaux.

Si en général, on étudie les fonctions pilotées de manière isolée ainsi que chacune des contraintes temporelles qui leur sont associées, la vérification des propriétés temporelles sur les segments correspondant au niveau applicatif et des segments correspondant à la transmission des messages sur les réseaux requiert d'étudier leur ordonnancement. Sur les calculateurs les plus évolués embarqués dans les véhicules, plusieurs centaines de runnables sont exécutés. De même, plusieurs milliers de signaux sous la forme de centaines de messages sont transmis sur les bus CAN. Ainsi, une fois que l'ensemble des runnables et l'ensemble des messages transmis sur les réseaux sont connus, il devient possible de valider deux autres types de segment.

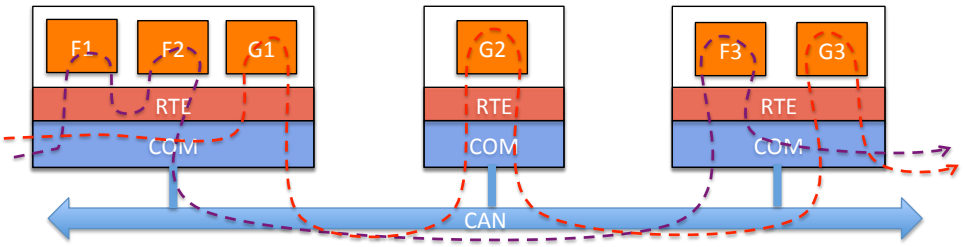

Figure 1.8 – Exemple de deux chaînes temporelles en concurrence sur des ressources partagées. L'obtention des temps de réponse des segments correspondant au niveau applicatif et des segments correspondant à la transmission des messages nécessite d'utiliser des approches d'ordonnancement.

1.2.6 Contraintes issues du processus de conception dans l'industrie automobile

Lors du processus de conception des fonctions pilotées par l'électronique embarquée dans un véhicule, les conceptions de l'architecture matérielle, des messageries réseaux (l'ensemble des paramètres des messages transmis sur les bus CAN) et des différentes fonctions logicielles sont réalisées en parallèle afin de raccourcir le temps de mise sur le marché d'un nouveau véhicule. Cela est rendu possible grâce à une forte réutilisation des composants existants d'une architecture matérielle sur l'autre, ainsi que grâce à l'utilisation de messagerie "souches" (génériques) qui sont ensuite adaptées à chaque configuration véhicule. Dans certains cas, la conception d'une nouvelle fonction peut nécessiter l'introduction d'un nouveau calculateur auquel cas une coopération plus rapprochée est mise en place entre les équipes impliquées. Cependant, dans les cas les plus courants, les concepteurs fonctionnels peuvent s'appuyer sur les composants et l'architecture existants.

Ce processus de conception est permis grâce à la séparation au niveau logiciel de la couche applicative et des "basic software components" en charge de la communication par la couche RTE. Ces deux niveaux fonctionnent de manière asynchrones ce qui permet de limiter l'impact des évolutions incrémentales qui peuvent être faites sur l'une au l'autre couche. La souplesse ainsi gagnée dans le processus de conception se paie en revanche au niveau des performances, car on ne contrôle pas cette transition. Lors de l'analyse des performances, il est en conséquent difficile de modéliser finement ce qu'il se passe au niveau du dernier type de segment qui fait l'interface entre la couche applicative et la couche responsable de la communication. Il n'est possible que de donner une valeur pire cas pour ces segments. Dans le cas typique d'un comportement périodique de la tâche logicielle responsable de la communication, le délai sera donc au plus égal à la somme de la période de cette tâche et de son temps d'exécution pire cas, cette situation survenant quand

l'information à transmettre est disponible juste après le début de l'exécution de la tâche la consommant.

La décomposition d'une contrainte de bout-en-bout en une chaîne temporelle linéaire est aussi source d'imprécisions. En effet, une description exhaustive des blocs (fonctionnels ou logiciels) nécessiterait de décrire le nombre important de données consommées et produites par chacun de ces blocs. Ainsi, au niveau logiciel, certaines variables partagées peuvent être utilisées par un grand nombre de runnables lors de leur exécution. Dans ces travaux, nous faisons l'hypothèse que les concepteurs sont capables d'identifier un chemin critique de données, entre le capteur et l'actionneur, pertinent pour la contrainte de bout-en-bout considérée. Dans le processus de conception, un grand nombre de variables utilisées pour les calculs mais non présentes sur les chemins étudiées ne sont pas considérées pour la vérification des contraintes temporelles. De fait, certains problèmes de dépendance entre runnables (à cause de variables partagées), de fraîcheur de donnée et de fréquence d'échantillonage ne sont pas couverts dans les travaux de cette thèse.

Une autre cause d'imprécision dans l'étude des contraintes temporelles est le cas des calculateurs de type "boîte noire". En effet, si le constructeur conserve l'exclusivité du développement de certains calculateurs critiques, la plupart des autres calculateurs sont achetés à des fournisseurs qui réaliseront l'intégration. Par exemple, le constructeur conserve généralement la responsabilité du contrôleur habitacle appelé chez PSA "Boîtier de Servitude Intelligent" , positionné de manière centrale et responsable de l'exécution d'un grand nombre de tâches en plus de faire passerelle entre les réseaux les plus importants. C'est aussi souvent le cas pour le calculateur en charge du contrôle moteur. En revanche, il est courant pour un constructeur d'acheter les calculateurs multimédia ou responsable de la correction de trajectoire chez un équipementier spécialisé. Il existe également un cas intermédiaire où le constructeur peut demander à un fournisseur d'intégrer des composants logiciels développés en interne auquel cas le constructeur peut avoir une idée plus précise sur le temps d'exécution du code confié au fournisseur. On parle alors de "boîte grise".

L'existence de ces boîtes noires, la conception parallèle et disjointe des applications et des réseaux ainsi que la complexité générale des architectures électroniques interdisent de pouvoir attaquer la vérification des contraintes de bout-en-bout de manière holistique, c'est à dire en considérant le système dans son ensemble. Pour ces raisons, une approche plus raisonnable consiste à décomposer les contraintes de bout-en-bout comme expliqué précédemment avant de travailler sur chacun des segments en procédant organe par organe, c'est à dire en étudiant successivement le comportement temporel au niveau des différents calculateurs "boîte blanche" et des réseaux. Alors que la problématique initiale consiste à suivre le chemin d'un signal de bout-en-bout, l'approche utilisée prend une direction orthogonale en étudiant localement, au niveau de chaque calculateur traversé et chaque réseau emprunté,

les interactions avec les autres signaux correspondant aux autres fonctions mises en oeuvre concurremment dans le véhicule .

1.3 Vérification des contraintes de bout-en-bout

Dans une architecture électronique automobile de type AUTOSAR, les contraintes de sûreté de fonctionnement des fonctions pilotées de criticité croissante (contrôle de la dynamique du véhicule) exigent une connaissance précise du comportement temporel du système afin notamment d'estimer de manière réaliste les temps de réponse de bout-en-bout. Ce problème est un enjeu de conception majeur sur lequel se mobilisent des acteurs industriels et académiques. Dans les travaux de thèse, nous nous situons dans le contexte de fonctions de niveau de criticité élevé pour lesquelles des garanties sur les performances temporelles de bout-en-bout sont requises. La connaissance des temps de réponse de bout-en-bout pourra alors être utilisée pour évaluer différentes solutions de conception, et ainsi optimiser l'architecture matérielle (choix du débit des réseaux, puissance des calculateurs) au regard des objectifs de performances et de sûreté de fonctionnement.

1.3.1 Techniques de vérification temporelle

On distingue classiquement trois approches de vérification temporelle adaptées pour les systèmes embarqués critiques :

- Le model-checking d'automates temporisés, qui fournit des résultats exacts mais, qui, généralement, ne passe pas à l'échelle sur des systèmes de taille industrielle. Au cours des 25 dernières années, des progrès significatifs ont été réalisés sur les algorithmes comme sur les outils, le logiciel en utilisation libre Uppaal [30] est par exemple aujourd'hui une référence dans le domaine. Néanmoins le model-checking souffre intrinsèquement du problème de l'explosion combinatoire de l'espace des états du système, et ne peut être a priori envisagé comme solution unique pour la vérification des systèmes de grande taille comme peuvent l'être ceux dans un véhicule où les calculateurs exécutent des centaines de runnables et où des centaines de messages sont échangés sur chaque bus.
- Le Calcul Réseau, ou Network Calculus, est une théorie développée pour déterminer des bornes supérieures sur les temps de transmission dans les réseaux [15, 16]. Le calcul réseau se base sur une algèbre particulière pour calculer et propager des contraintes exprimées sous forme d'enveloppes. Ces enveloppes définissent la quantité de travail à réaliser au cours du temps (ex : pour exécuter une tâche de façon périodique) et la quantité de travail offerte par les ressources (puissance de calcul d'un processeur ou débit

d'un réseau). Le calcul réseau offre plusieurs avantages par rapport aux analyses d'ordonnançabilité traditionnelles. En premier lieu, il repose sur des fondements mathématiques qui permettent de réaliser les calculs de façon algébrique, donc sans recourir à des analyses ad-hoc dont la validité est plus difficile à vérifier. D'autre part, le calcul réseau passe très bien à l'échelle et est adapté pour des systèmes de très grande taille (plusieurs dizaines de milliers de flux de données échangés par plusieurs centaines de calculateurs). Pour ces raisons, le calcul réseau est utilisé en certification dans l'avionique depuis une dizaine d'année (par exemple sur l'A380). Un de ses désavantages est d'être généralement plus pessimiste que le model-checking et que les analyses d'ordonnançabilité traditionnelles, et donc potentiellement de conduire à un surdimensionnement des ressources matérielles. Si depuis, des progrès théoriques sont réalisés et les différences avec les autres approches s'amenuisent [39], cette approche est moins pertinente pour notre problèmatique car nous étudions individuellement les différents calculateurs et réseaux. Le périmètre des systèmes étudiées (un calculateur ou un réseau CAN) ne justifie pas d'employer le calcul réseau afin de faire face à la taille du système.

– L'ordonnancement temps réel est une discipline très active depuis le début des années 70 avec un grand nombre de résultats adaptés à des contextes technologiques variés. L'approche est ici, non pas d'explorer exhaustivement l'espace des états du système, mais de construire et d'analyser un sous-ensemble de trajectoires que l'on peut prouver défavorables d'un point de vue temporel. La plupart des résultats concernent l'ordonnancement des processeurs mais des extensions ont été rapidement réalisées, par exemple l'approche holistique développée à York dans [31], pour les systèmes distribués puis les approches à base de trajectoire [43, 41]. Plus récemment, des avancées ont été faites vers des modèles compositionnels, que ce soit dans le cas centralisé [27] ou dans le contexte de systèmes distribués [65] ou encore l'approche développée à la T.U. Braunschweig, utilisée dans le logiciel commercial Symta/S [26].

Dans le contexte de la vérification des contraintes de bout-en-bout lors de la conception d'une fonction pilotée pour une voiture, cette dernière approche est plus adaptée au regard de la complexité du système et du choix fait d'étudier séparément les temps de réponses sur les calculateurs et les réseaux.

1.3.2 Contributions

En raison des contraintes liées au processus de conception considéré dans le cadre de cette thèse, nous avons fait le choix de passer d'une étude globale à une étude basée sur une décomposition hiérarchique. Ainsi, nous contribuons séparément à la vérification des contraintes de bout-en-bout au niveau de l'ordonnancement des composants logiciels sur les calculateurs et des messages sur les réseaux

CAN. En effet, la présence de boîtes noires et le développement en parallèle, du matériel, du logiciel et des réseaux rend impossible la validation des contraintes en les considérant de bout-en-bout. De plus, l'indisponibilité d'un certain nombre d'information ainsi que l'impossibilité de saisir toutes les interactions entre les fonctions dû à la complexité générale de l'ensemble de l'architecture électronique d'une voiture rend stérile toute tentative d'optimisation globale des performances temporelles du système.

Les contributions de cette thèse sont présentées dans les trois chapitres suivants et correspondent respectivement aux questions suivantes.

Comment garantir un respect des contraintes de temps sur les calculateurs AutoSar en optimisant leur utilisation ? La grande majorité des composants logiciels et des messages émis ont un comportement de type périodique avec échéance sur requête (leur échéance est égale à la date de leur prochaine activation), entre autres raisons à cause du grand nombre de fonctions de contrôle-commande. Dans cette mesure, des améliorations sur les comportements temporels pire cas des différents composants peuvent être obtenues via un effort de lissage de la charge périodique sur les différents calculateurs et réseaux en utilisant des offsets pour décaler leurs dates d'activation. L'application de cette approche sur les réseaux CAN [24] a permis de réduire considérablement les temps de réponse des messages et permet d'utiliser de manière plus efficace la puissance de calcul des processeurs ou la bande passante des réseaux. Ceci est très intéressant du point de vue industriel car cela permet d'être robuste vis-à-vis de la charge apériodique critique, de monter plus haut en charge sur ces différents organes et de pouvoir ajouter de nouvelles fonctionnalités sur une architecture existante sans changer le support matériel. Par ailleurs, de plus en plus de calculateurs multi-cœurs sont intégrés dans les véhicules pour monter en puissance alors que les montées en fréquence des processeurs commence à montrer leurs limites en fiabilité. Dans le chapitre 2, nous démontrons comment appliquer cette technique sur des calculateurs multi-cœurs.

Comment étudier le comportement temporel des messages sur un réseau CAN en modélisant les dérives d'horloge sur les stations émettrices ? Nous cherchons à modéliser le phénomène de dérive d'horloge et étudier leur influence sur les distributions de temps de réponse obtenues par simulation. Par opposition aux temps de réponse pire cas qui étaient les seules informations que les concepteurs pouvaient typiquement obtenir par analyse, les distributions de temps de réponses sont incomparablement plus riches en information. Leur connaissance peut permettre entre autre de dimensionner l'architecture matérielle plus finement en utilisant des critères quantitatifs basés sur les probabilités de défaillance (par exemple, défaillances par heure) dans le contexte où l'on associe le dépassement du temps de réponse d'un message par rapport à une valeur seuil (l'échéance) comme

une faute. Cela permet alors de considérer des temps de réponses sensiblement plus faibles en comparaison des valeurs pire cas traditionnellement utilisées. Dans le chapitre 3, nous étudions comment conduire des simulations avec dérives d'horloge pour obtenir des résultats pertinents pour différents horizons de simulation : des courtes simulations pour étudier ce qu'il se passe localement dans des scénarios pire cas et des longues simulations pour étudier le comportement sur le long terme (durée de vie du véhicule) des messageries CAN.

Comment obtenir par analyse une estimation de la distribution de temps de réponse d'une trame CAN ? Toutes les trames sur un réseau n'ont pas le même niveau de criticité. Il est donc important d'avoir un modèle fin du comportement donné d'une trame critique, généralement de haute priorité, et d'exhiber des évaluations de ses temps de réponse. Pour traiter ce problème, nous avons adopté une approche probabiliste qui permet d'avoir une estimation de la distribution de temps de réponse d'une trame émise sur un réseau CAN. Nous calculons des estimations car la complexité des calculs augmente considérablement au fur et à mesure que sont étudiés des messages de moins en moins prioritaires. En revanche, si certaines approximations sont tout le temps appliquées, l'approche que nous présentons dans le chapitre 4 incorpore plusieurs paramètres contrôlés par l'utilisateur permettant de jouer sur la précision des résultats obtenus.

Annexes. Ce mémoire comporte également en annexe deux autres contributions réalisées dans le cadre de cette thèse. Ces contributions s'inscrivent dans le contexte industriel de la thèse et nous avons fait le choix de les séparer du corps du mémoire pour lequel nous nous sommes restreint aux travaux de recherche. L'annexe B décrit l'évaluation d' "aiT" de la société AbsInt, un outil logiciel permettant d'estimer des bornes sur les pire-temps d'exécution de composants logiciels. Cette étude est en lien avec les travaux de chapitre 2 pour lesquels la connaissance précise des temps d'exécution des runnables est importante. En effet, il est de pratique courante d'utiliser une évaluation "gros grain" basée sur le retour d'expérience et l'expertise des concepteurs des fonctions, par exemple sous la forme d'une valeur de 1 à 10 représentant grossièrement le temps d'exécution d'un runnable. Utiliser un outil d'analyse statique du code permet effectivement d'avoir une connaissance plus fine des temps d'exécution relatifs des différents composants logiciels qui seront exécutés sur les calculateurs avant le développement du matériel. Enfin, l'annexe C comporte la description de dispositifs intégrés aux contrôleurs de communication visant à détecter et résoudre les situations d'encombrement des réseaux CAN. Ces travaux ont été réalisés suite à des observations faites sur les résultats présentés dans le chapitre 3 et ont donné lieu à trois brevets [7, 8, 9].

Chapitre 2

Ordonnancement de runnables sur les calculateurs AutoSar

Résumé Dans ce chapitre nous nous intéressons à l'analyse temporelle des calculateurs embarqués dans les voitures. En utilisant les hypothèses compatibles avec AutoSar, nous étudions comment lisser la charge induite par les composants logiciels exécutés sur les calculateurs afin d'optimiser l'utilisation de la capacité de leurs cœurs. Pour cela, sur la base d'un ordonnancement statique cyclique, nous cherchons à calculer les valeurs des offsets entre les runnables dans l'objectif d'éviter les pics de charge ainsi que de respecter les contraintes d'échéances. Il s'agit d'un problème complexe pour lequel nous proposons des heuristiques car une étude exhaustive des possibilités n'est pas possible en pratique. Nous proposons donc une méthode de construction guidée par la garantie des contraintes de temps. Dans certaines conditions, nous présentons également des garanties sur les performances des algorithmes proposés pour lisser la charge. Au fur et à mesure, nous relaxons les hypothèses de travail jusqu'à proposer une approche permettant de traiter le cas général où les runnables sont ordonnancés sur un même cœur par plusieurs tâches séquenceur distinctes.

Sommaire

2.1 Conception des calculateurs embarqués dans l'automobile

2.1.1 Contexte

Boîtes blanches et boîtes noires Dans ce chapitre, nous nous intéressons aux calculateurs embarqués dans les voitures et à la vérification du respect des contraintes de temps par les activités déployées sur ceux-ci. Du point de vue d'un constructeur automobile, il convient de distinguer deux types de calculateurs dans l'architecture électronique embarquée : les calculateurs "boîtes blanches", sur lesquels le constructeur automobile a une connaissance complète de l'architecture et des propriétés temporelles, et les calculateurs "boîtes noires" commandés chez les fournisseurs pour lesquels ne sont connus que les entrées et sorties ainsi que des garanties sur les temps de réponses pour les tâches dont ils sont responsables. Par exemple, le calculateur principal du domaine habitacle ou le contrôleur en charge du contrôle moteur seront en général des boîtes blanches tandis que des boîtiers dédiés à des fonctions très spécifiques comme l'ABS ou l'EPS peuvent être des boîtes noires achetés chez des fournisseurs. Dans le contexte du processus de développement chez un constructeur automobile, les propriétés temporelles des fonctions exécutées sur les boîtes noires ne peuvent que faire l'objet de spécifications écrites dans un cahier des charges. En revanche, les calculateurs "boîtes blanches" sont développés spécifiquement chez le constructeur car ils sont stratégiques, relèvent des avantages compétitifs et permettent de se différencier de la concurrence. Ainsi, nous ne nous intéressons dans ce chapitre qu'aux cas des boîtes blanches.

Développement multi-source Un nombre important de tâches sont exécutées sur ces calculateurs opérés par un OS temps-réel de type AutoSar (ou du moins OSEK/VDX). De plus, même si le développement et l'intégration de ces calculateurs sont réalisés en interne, le standard AutoSar et les contraintes économiques encouragent à utiliser du code développé en externe, acheté à des fournisseurs. On parle de "développement multi-source". Une des raisons principales de ce phénomène est que cette pratique permet de réduire le nombre d'ECUs dans les voitures alors que celui-ci dépasse 70 dans les modèles de véhicules haut de gamme. Un des résultats majeurs de l'initiative AutoSar, est de permettre aux constructeurs de passer du paradigme "une fonction par calculateur" à des architectures plus centralisées où plusieurs fonctions peuvent être intégrées au sein d'un même ECU grâce aux mécanismes de protection appropriés. Ceci est permis par la standardisation des briques logicielles dans AutoSar ainsi que par les mécanismes de protection d'AutoSar OS [52]. En effet, intégrer du code provenant de plusieurs sources soulève des problèmes de responsabilité en cas de défaillance du système. Il est donc important d'isoler autant que possible les composants logiciels provenant de différentes sources

à la fois pour limiter les problèmes pouvant résulter d'interférences non-identifiées à la conception ainsi que pour pouvoir gérer le partage des responsabilités.

Calculateurs multi-cœurs De plus, l'électronique embarquée dans les véhicules est en train de vivre un autre tournant. Comme dans l'informatique grand public auparavant, nous avons atteint un point où il ne devient plus possible de répondre aux exigences de performances croissantes en augmentant la fréquence de fonctionnement des processeurs des calculateurs. Ainsi, à l'heure où les besoins en puissance de calcul deviennent de plus en plus importants, les constructeurs automobiles et leurs fournisseurs ont recours au même remède et introduisent progressivement des calculateurs multi-cœurs dans l'architecture électronique embarquée des véhicules. Ces plateformes multi-cœurs ouvrent également la voie à de nouvelles approches dans l'électronique embarquée. Ainsi, la possibilité d'exécuter en parallèle les tâches permet de satisfaire plus facilement les exigences de sûreté telle qu'introduites par l'ISO 26262 ainsi que de mettre en œuvre de nouveaux cas d'usage dans le domaine des systèmes embarqués dans l'automobile (décrits ci-après en 2.1.2). En contrepartie, ces possibilités rendent la conception, le développement et la vérification des applications logicielles plus compliquées. Ainsi, les constructeurs et les fournisseurs ont donc besoin de nouveaux outils et méthodologies pour intégrer et valider ces solutions multi-cœurs. A présent, l'enjeu majeur est d'adapter les méthodes de conception existantes aux nouvelles contraintes des architectures multi-cœurs. L'ordonnancement des composants logiciels est donc au cœur de cette problématique et nécessite d'être re-étudiée.

2.1.2 Principaux cas usages des processeurs multi-cœurs

Il existe une grande variété d'architectures matérielles et logicielles pour les calculateurs multi-cœurs. Par exemple, en ce qui concerne seulement la partie matérielle, les fournisseurs proposent différentes approches : des cœurs identiques, des cœurs hétérogènes fonctionnant à des fréquences différentes et/ou avec différents jeux d'instructions, et toute une variété d'entrées/sorties ou d'organisations différentes de la mémoire. Cependant, les fondeurs produisent des processeurs multi-cœurs depuis un certain nombre d'années pour le marché de l'ordinateur personnel ce qui pourrait bien influencer l'industrie automobile car ces processeurs de PC et leurs architectures ont démontré leur utilisabilité et seront probablement moins chers du fait de leur production à grande échelle.

L'introduction des calculateurs multi-cœurs dans l'architecture électronique des véhicules n'en est qu'à ses débuts mais ouvre des possibilités intéressantes. Dans cette partie, nous discutons succinctement les utilisations pertinentes des calculateurs multi-cœurs dans un véhicule ainsi que leurs solutions d'implémentations appropriées.

Simplifier l'architecture électronique embarquée dans les véhicules

Les gains de performances offerts par les architectures multi-cœurs offrent la possibilité de simplifier l'architecture électronique des véhicules : il devient possible d'exécuter les fonctions préalablement exécutées sur plusieurs calculateurs distincts sur plusieurs cœurs au sein d'un seul et unique calculateur . Cette évolution vers des architectures plus centralisées permet, de plus, aux constructeurs de réduire le nombre de connexions réseaux et simplifier les bus de communication. Cela revient à transférer la complexité depuis l'architecture opérationnelle à l'échelle des véhicules vers l'architecture matérielle et logicielle à l'échelle des calculateurs. De plus, l'ordonnancement statique cyclique permet d'ajouter facilement de nouvelles fonctions (de nouveaux runnables) sur un calculateur existant. Cependant, il est difficile en pratique d'opérer des changements en rupture au niveau des architectures embarquées du fait de la reconduction d'un certain nombre de calculateurs et réseaux, une pratique courante, en particulier chez les constructeurs généralistes. Il est donc difficile de prédire dans quelle mesure ces architectures seront adoptées dans le futur.

Exécuter des applications nécessitant une importante puissance de calcul

Les calculateurs multi-cœurs permettent de répondre directement à des besoins de plus en plus importants en puissance de calcul. Ce sont ceux, par exemple, des applications relatives au contrôle moteur ou au traitement d'image. Cet usage n'impose pas de contrainte sur l'architecture matérielle et une architecture avec des cœurs identiques est donc facilement envisageable. Pour ces applications, il est de plus possible de profiter de la parallélisation des traitements sur une architecture multi-cœur. Typiquement, en traitement d'image, il est courant d'exécuter des copies du même composant logiciel sur les différents cœurs afin de traiter en parallèle différentes sous-parties de l'ensemble des données d'entrée.

Améliorer la sûreté de fonctionnement

Dans le domaine de la sûreté de fonctionnement, les nouvelles possibilités ouvertes par l'adoption de calculateurs multi-cœurs permettent d'envisager des mécanismes répondants aux exigences de ce domaine. En particulier, nous avons identifié trois moyens de tirer partie de l'architecture multi-cœur pour augmenter la sûreté de fonctionnement.

Isolation La première idée consiste à séparer le code sûr ("trusted") dans lequel on a confianc, du code non-sûr ("non-trusted"), dans lequel on a pas confiance en les exécutant sur des cœurs différents. En effet, les constructeurs peuvent considérer le code provenant d'un fournisseur comme "non-trusted" tout comme un fournisseur

en charge de l'intégration finale d'un calculateur peut considérer le code fourni par le constructeur comme "non-trusted" pour des questions de responsabilité. Ce type d'isolation entre composants logiciels nécessite cependant des mécanismes de protection mémoire, protection temporelle et protection des ressources tels que fournis dans AutoSar OS [52] ou comme pourrait le permettre une approche à base de machines virtuelles [49].

Redondance La seconde méthode consiste à faire de la redondance active en exécutant les tâches critiques en parallèle, avec éventuellement un système de vote choisissant le résultat donné par la majorité des runnables dupliqués. Il est envisageable de dupliquer soit la totalité des composants logiciels exécutés par un cœur sur un autre cœur, soit seulement les runnables les plus critiques dans l'objectif de trouver un compromis entre la sûreté et les besoins en puissance de calcul. Afin d'augmenter encore plus le niveau de sûreté, il est possible d'utiliser une approche du type "N-Version Programming" (NVP) [14] qui consiste à employer différentes versions d'un même composant logiciel développées par différents équipementiers au lieu d'utiliser plusieurs copies identiques du runnable en question.

Autosurveillance Enfin, les architectures multi-cœurs permettent d'implémenter facilement des fonctions de surveillance ("monitoring"). Dans ce cas, un cœur peut être chargé de surveiller la bonne exécution de fonctions situées sur un autre cœur. De plus, il est possible d'augmenter le niveau de sûreté en utilisant plusieurs processeurs au lieu de plusieurs cœurs au sein du même processeur, quoique ces solutions soient en conséquence plus chères à mettre en œuvre.

Usage spécialisé d'un cœur

Un dernier cas d'usage pertinent d'une architecture multi-cœurs consiste à utiliser un des cœurs pour s'occuper spécifiquement de certains services système de bas niveau. Dans le contexte d'AutoSar, un cœur pourrait être en charge de toutes les entrées/sorties, de toutes les couches de communication et des "basic softwares" tandis qu'un autre cœur serait uniquement en charge des composants logiciels de haut niveau (le code des applications). Alternativement, un cœur pourrait être dédié à l'exécution des composants logiciels périodiques tandis qu'un autre cœur serait dédié à la gestion des interruptions et des composants logiciels événementiels comme réalisé dans le projet PharOS [10] sur un microcontrôleur SX12 Freescale.

2.1.3 Travaux existants

Dans le domaine des architectures multi-cœurs, il existe deux approches de partitionnement des composants logiciels : l'allocation statique de ceux-ci sur les

différents cœurs ou des mécanismes d'allocation dynamique (ordonnancement multiprocesseur "global") qui équilibrent la charge de travail au cours de l'exécution. Cette dernière met en jeu des interactions complexes au niveau des différentes tâches et ressources dont le fonctionnement devient alors plus difficile à prédire et à valider. Pour cette raison, dans le domaine automobile, les concepteurs favorisent l'approche de partitionnement statique et l'utilisation de mécanismes déterministes comme l'ordonnancement statique cyclique, conformément au choix fait par le consortium AutoSar [11]. L'approche de partitionnement statique a déjà été largement traitée dans d'autres travaux comme par exemple dans [13] et [32, 37, 58, 59]. Cependant, au niveau de chaque cœur, ces travaux utilisaient des algorithmes d'ordonnancement à priorité tels FPP ou EDF et non une approche d'ordonnancement statique cyclique des tâches qui est calculée hors-ligne et exécutée de façon répétitive. Les algorithmes de configuration présentés dans ce chapitre sont proches de [22] (ordonnancement monoprocesseur avec offsets) et [24] (ordonnancement de trames avec offsets) mais nous considérons pour notre part une plateforme multi-cœurs. Enfin, les travaux sur la périodicité stricte [36], continués dans [40] après la publication de nos travaux, s'intéresse à un problème presque identique. Cependant, dans notre cas, nous faisons le choix d'être moins stricte au niveau de la gigue des runnables dont les dates de début d'exécution ne sont pas nécessairement strictement périodiques, mais les dates d'activation le sont (voir Figure 2.1).

Enfin, il existe dans l'industrie des outils internes chez les constructeurs et fournisseurs ainsi que des outils commerciaux comme RTaW Netcar-ECU [62], Symta/S de Symtavision [66] et les outils de la société Inchron [29] qui ont été développés pour l'ordonnancement. Cependant, les algorithmes utilisés sont propriétaires et sont souvent utilisés pour des cas très spécifiques.

2.1.4 Contributions

Dans ce chapitre, nous présentons une approche adaptée aux calculateurs AutoSar pour étudier et améliorer le comportement temporel des tâches qui y sont exécutées. Nous nous intéressons donc au cas des calculateurs "boîte blanches" multicœur où sont ordonnancés de nombreux composants logiciels provenant éventuellement de plusieurs sources et correspondant à différents niveaux fixes de priorité. L'adoption progressive de ces approches multi-sources (pour le logiciel) et multicœur (pour le matériel) induit des changements dans l'architecture des calculateurs et le processus de conception. Ainsi, lors du développement du calculateur, il est nécessaire de définir l'ordonnancement de plusieurs centaines de ces composants logiciels afin de respecter et vérifier les contraintes temporelles. De manière similaire à ce qui a été fait sur CAN [24], nous chercherons à décaler les dates d'arrivées des runnables afin de diminuer les temps de réponses pour les runnables les moins

prioritaires. Dans le cadre de cas travaux, nous faisons l'hypothèse d'un ordonnancement préemptif à priorité fixes conformément aux pratiques dans l'industrie automobile.

Dans la pratique, il y a amplement plus de runnables que le nombre de tâches supportées par les systèmes d'exploitation des calculateurs embarqués [1]. Pour cette raison, les runnables sont regroupés et ordonnancés au sein d'une ou plusieurs "tâches séquenceur". En accord avec la pratique dans l'industrie automobile, nous faisons le choix de réaliser un ordonnancement statique cyclique de ces runnables au sein de ces tâches séquenceurs. Nous faisons également l'hypothèse d'un partitionnement statique des runnables sur les différents cœurs car cette l'approche a plus de chance d'être adoptée dans un premier temps. Cette approche de partitionnement statique et d'ordonnancement cyclique des runnables au sein de tâches préemptives à priorités fixes est plus simple à mettre en œuvre et plus prédictible pour les concepteurs en comparaison avec une approches d'allocation dynamique (ordonnancement global) des runnables sur les cœurs et d'ordonnancement dynamique des runnables ou des tâches séquenceurs sur chacun des cœurs (Earliest Deadline First par exemple).

L'objectif des travaux présentés dans ce chapitre est de proposer des algorithmes réellement utilisables en pratique pour partitionner et ordonnancer un ensemble de runnables sur les différents cœurs dans le respect des contraintes de conception et en essayant d'uniformiser la charge de calcul dans le temps. La qualité du lissage de la charge est un enjeu important. Un lissage optimal permet de faciliter le respect des échéances en diminuant les temps de réponses des tâches moins prioritaires et permet aussi à terme de réduire les coûts du matériel (les pics de charge étant moins forts et moins fréquents) ainsi que de faciliter l'ajout de nouvelles fonctions dans une approche de développement incrémental comme cela est souvent le cas dans ce domaine.

Comme nous faisons le choix d'un partitionnement statique et de l'utilisation de tâches séquenceurs pour ordonnancer les runnables, ce problème général d'ordonnancement peut se scinder en deux sous-problèmes distincts :

- partitionner l'ensemble des runnables sur les différents cœurs
- ordonnancer chacun des cœurs via la construction des tâches séquenceur

Ces deux sous-problèmes sont des problèmes d'optimisation. Le premier consiste à allouer les runnables sur les différents cœurs dans l'objectif d'équilibrer au mieux la charge sur les cœurs. Le second problème consiste à choisir les offsets initiaux des runnables dans le but de minimiser les pics de charge. En décrivant leur complexité, nous montrons que ces deux problèmes ne peuvent pas être résolus de manière optimale en pratique. Nous présentons alors des heuristiques simples pour résoudre progressivement chacun de ces problèmes. Le problème de l'ordonnancement sur

1. En fonction de la classe de conformance, OSEK/VDX et AutoSar OS impose seulement un minimum de 8 à 16 tâches. Il n'est donc pas garanti d'en avoir plus à disposition.

des cœurs est lui-même étudié en deux temps. Nous étudions d'abord comment lisser la charge dans le cas où il n'y a qu'une seule tâche séquenceur présente sur le cœur étudié avant d'étendre l'approche au cas général où plusieurs tâches dont des tâches séquenceurs peuvent être présentes et sont ordonnancés de manière préemptives et selon des valeurs de priorité fixes.

En lien avec les travaux contributions présentées dans ce chapitre, le lecteur trouvera également en Annexe B l'évaluation d'un outil de calcul des pires temps d'exécution des runnables. Nous faisons l'hypothèse dans ce chapitre que nous avons été capable d'obtenir les valeurs de temps d'exécution des runnables mais ce qui n'est pas toujours possible en réalité (en particulier lors du développement du nouvelle fonction). Il est donc pratique courante d'utiliser des métriques "gros grain" pour évaluer le temps d'exécution des runnables comme par exemple une valeur comprise entre 1 et 10. Le résultat des algorithmes de lissage de la charge périodique présentés dans ce chapitre est certainement meilleur si les valeurs de pire temps d'exécution sont connues précisément. Néanmoins, il est tout à fait possible d'utiliser les résultats présentés dans ce chapitre en utilisant des métriques plus grossières tant qu'elles sont proportionnelles aux temps d'exécution afin de pouvoir calculer des valeurs de charge pour effectuer le lissage.

2.2 Modèle du calculateur

Dans ce chapitre, nous considérons un grand nombre n de composants logiciels élémentaires périodiques, aussi dénommés runnables, que nous cherchons à exécuter sur un calculateur composé de m cœurs. En pratique, ce système est mis en œuvre en procédant à des appels de fonctions vers les runnables dans le corps du code de la tâche OS qui sert à gérer le séquencement (la tâche séquenceur).

2.2.1 Caractéristiques des runnables

Les runnables ont des échéances sur requête, c'est-à-dire que leur exécution doit être terminée avant leur prochaine activation. Le $i$ème runnable est exprimé ainsi : $\mathscr{R}_i = (C_i, T_i, O_i, P_i, \{\mathscr{R}\}_i, K_i)$ avec :

- C_i le pire temps d'exécution (WCET) du runnable
- T_i la période du runnable
- O_i l'offset initial du runnable.
- P_i le niveau de priorité fixe du runnable utilisé par l'ordonnanceur
- $\{\mathscr{R}_c\}_i$ la contrainte de co-localisation
- K_i la contrainte de localisation

L'offset initial désigne l'instant auquel la première instance du runnable est activée, les instances suivantes seront déclenchées périodiquement à partir de ce moment. Ainsi, le choix de cet offset influe directement la répartition de la charge de travail.

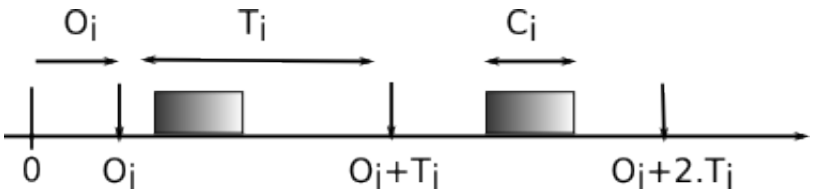

Figure 2.1 – Modèle des runnable. Après son activation, l'exécution d'une instance d'un runnable commence après une durée variable et doit avoir fini d'être exécutée avant l'arrivée de sa prochaine instance (c.à.d. l'échéance est égale à la période d'un runnable)

Les runnables qui auront le même niveau de priorité sur un cœur seront regroupés au sein de la même tâche séquenceur. Cela permet de séparer les runnables mettant en oeuvre des fonctions de niveaux de criticité différents. Dans une certaine mesure, cela permet également la ségrégation de runnables provenant de différentes sources.

Ensuite, $\{\mathscr{R}_c\}_i$ sert à grouper les runnables inter-dépendants. C'est une liste des runnables devant être exécutés sur le même cœur que $\mathscr{R}_i$. En effet, certaines caractéristiques des composants logiciels, comme l'usage de variables partagées, peuvent nécessiter l'allocation de certains runnables sur un même cœur.

De plus, certaines spécificités de l'architecture, comme certaines entrées/sorties localisées au niveau d'un cœur particulier, peuvent nécessiter d'allouer certains runnables sur un cœur particulier. Cette contrainte de "localisation" est exprimée par K_i qui identifie, le cas échéant, le cœur où le runnable doit être exécuté.

Nous faisons l'hypothèse que les valeurs de $\{\mathscr{R}_c\}_i$ et K_i sont cohérentes, c'est-à-dire qu'il n'y a pas deux runnables liés par une contrainte de co-localisation et dont les contraintes de localisation sont différentes.

2.2.2 Tâches séquenceurs

Les runnables sont ordonnancés sur leur cœur respectif par l'intermédiaire de tâches séquenceurs. On associe à celles-ci des tables d'ordonnancement décomposées en "slots" (créneaux) où sont stockées les dates d'activation des différents runnables afin de les exécuter au moment opportun. Chaque slot de la table est ainsi associé à un ensemble de runnables pour lesquels une instance sera activée à la date d'activation du slot. Une fois arrivée à la fin de cette table, on recommence à partir du premier slot et ceci de manière cyclique. L'allocation des runnables dans les slots est une conséquence directe du choix de l'offset initial des runnables (résultat des algorithmes présentés dans ce chapitre) et de la période de ces runnables.

La tâche séquenceur de priorité P ordonnancée sur le cœur K est exprimée ainsi : $S_{p,k} = (T_{tic}, T_{cycle}, \Omega(\{R\}_{p,k})$ avec :

- T_{tic} la durée élémentaire d'un "slot"
- T_{cycle} la durée totale de la tâche séquenceur
- $\Omega(\{R\}_{p,k}$ est la table d'ordonnancement des runnables de priorité P sur le cœur K

Par exemple, dans l'automobile, ces valeurs sont typiquement $T_{cycle} = 1000\,ms$ et $T_{tic} = 5\,ms$. Il est important de noter que T_{cycle} doit être un multiple du PPCM des périodes des runnables et que le PGCD des périodes des runnables doit être un multiple de T_{tic}. En conséquence, il y a T_{cycle}/T_{tic} slots dans la table de la tâche séquenceur. Ainsi, pour établir l'ordonnancement des runnables, nous cherchons à construire cette table pour chaque tâche séquenceur. Choisir l'offset initial O_i d'un runnable $\mathscr{R}_i$ consiste à la placer dans le slot à la $\frac{O_i}{T_{tic}}$-ème place de la table. Ensuite, il sera nécessaire de placer ce runnable dans la table tous les $\frac{T_i}{T_{tic}}$ slots afin de respecter sa périodicité. Des algorithmes de calcul des offsets sont décrits en section 2.4.2 et section 2.4.3.

La figure 2.2 montre le séquencement correspondant à l'ensemble de runnables $\mathscr{R}_i(T_i, C_i)$ suivants : $\mathscr{R}_1(10,2)$, $\mathscr{R}_2(10,1)$, $\mathscr{R}_3(20,3)$ et $\mathscr{R}_4(20,2)$. Pour $T_{tic} = 5$ et $T_{cycle} = 40$, cela donne une table d'ordonnancement avec 8 slots et séquencée de la manière suivante :

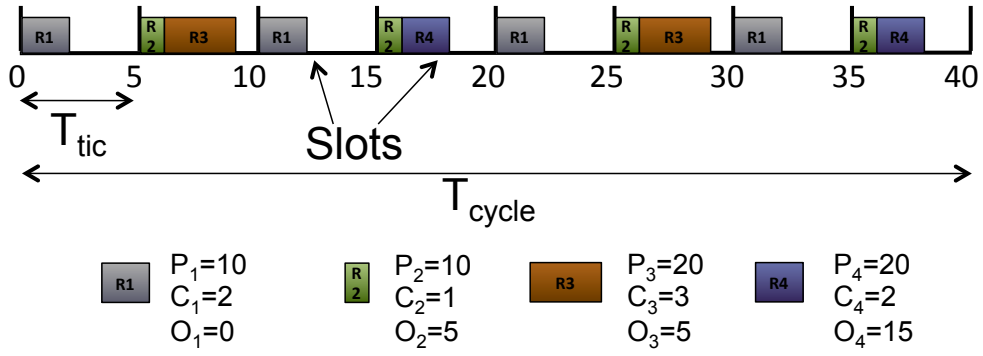

Figure 2.2 – Modèle de tâche séquenceur

2.2.3 Hypothèses de travail

Dans le contexte de nos travaux, nous posons certaines hypothèses de travail, qui, de notre expérience, sont généralement valides dans le domaine de l'électronique embarquée dans les véhicules :

- Chaque runnable est activé périodiquement. En conséquence, l'ordonnancement du système est prévisible et est une conséquence directe des dates des premières activation des runnables (*i.e.*, leurs offsets initiaux).
- A la conception, il est possible de choisir librement les offsets des runnables (voir [22]), dans la limite de leur période : $O_i < T_i$. Ces offsets seront assignés pendant la construction de la table d'ordonnancement de la tâche séquenceur dans le but d'uniformiser la charge du cœur pendant un cycle d'ordonnancement.
- Les pire temps d'exécution des runnables sont petits par rapport à T_{tic}. Des valeurs courantes pour un cas d'application réaliste sont $T_{tic} = 5\,ms$ et $C_i \leq 300\,\mu s$.
- Tous les cœurs du calculateurs ont la même fréquence de fonctionnement.
- Les runnables alloués sur les différents cœurs sont indépendants et peuvent être ordonnancés indépendamment. Ainsi, il est possible d'ordonnancer indépendamment chaque cœur. Cette hypothèse concorde avec les choix faits pour AutoSar pour les architectures multi-cœurs [11].

Cette dernière hypothèse permet de diviser le problème initial en deux sous-problèmes distincts. Le premier sous-problème consiste donc à réaliser le partitionnement de l'ensemble des n runnables sur les m cœurs en respectant les contraintes de co-localisation et de localisation tout en cherchant à équilibrer la charge entre les différents cœurs. Le second sous-problème consiste à construire la table d'ordonnancement des runnables pour chacune des tâches séquenceurs.

2.2.4 Conditions d'ordonnançabilité

Dans le contexte étudié ici, le système résultant est considéré ordonnançable, et pourra donc être déployé, si et seulement si les conditions suivantes sont vérifiées sur chacun des cœurs :

- Les périodes d'activation des runnables sont strictement respectées dans l'ordonnancement résultant.
- Les offsets initiaux des runnables sont strictement plus petits que leurs périodes respectives.
- Un runnable doit avoir fini son exécution avant l'arrivée de sa prochaine instance.

Implicitement, la dernière condition indique qu'il est fait l'hypothèse que les échéances sont égales à la période des tâches. Ceci est un choix courant et ne limite pas les résultats présentés ci-après. Il est tout à fait possible de relaxer cette hypothèse et d'utiliser des échéances plus courtes par la suite.

2.3 Partitionnement des runnables

Dans cette section, nous proposons un algorithme de partitionnement statique d'un ensemble de runnables sur un nombre fixé de cœurs (pour une application pratique). Dans cette étape nous distribuons les runnables sur les cœurs sans vérifier les conditions d'ordonnancement, ce qui ne sera possible qu'après la construction des tâches séquenceurs. L'enjeu est de distribuer le plus équitablement possible la charge sur les cœurs du calculateur tout en assurant le respect des contraintes d'allocation (localisation et co-localisation).

Nous faisons le choix de ne pas tenir compte de la priorité des runnables dans cette étape. Une approche possible consisterait à distribuer les runnables sur les cœurs de manière à limiter le nombre de niveaux de priorité présents sur chaque cœur et donc le nombre de tâches séquenceurs présentes sur chaque cœur afin de limiter le nombre de préemptions lors de l'ordonnancement. Nous considérons que cela relève des choix de conception. Néanmoins, cela est tout à fait possible à réaliser en jouant avec les contraintes d'allocations définies pour les runnables. Par exemple, le concepteur pourra définir des contraintes de co-localisation pour les ensembles de runnables de même priorité.

2.3.1 Enoncé du problème

Nous cherchons à déterminer $\mathscr{P}(\{R\}, m)$, le partionnement final de l'ensemble de tous les runnables $\{R\}$ sur m cœurs, sous les contraintes de localisation et de co-localisation exprimées pour chacun des runnables, dans l'objectif de minimiser le maximum de charge allouée sur les différents cœurs.

Allouer n runnables sur m cœurs revient à partitionner un ensemble de n éléments en m sous-ensembles. Par définition, le nombre de possibilités pour ce problème est donné par le nombre de Stirling de second ordre [6] :

$$\frac{1}{m!}\sum_{i=0}^{m}(-1)^{(m-i)}\binom{m}{i}i^n$$

De plus, il faut également distinguer les différents cœurs du fait des éventuelles contraintes de localisation et donc considérer les $m!$ combinaisons possibles. En conséquence, nous obtenons donc jusqu'à $\sum(-1)^{(m-i)}\binom{m}{i}i^n$ possibilités uniquement pour ce problème de partitionnement. Un tel nombre interdit toute recherche exhaustive des solutions même pour un petit nombre de runnables. Par exemple, pour 30 runnables et 2 cœurs, l'espace de recherche contient déjà plus d'un milliard de possibilités distinctes.

Au vu de la complexité, nous proposons donc d'équilibrer la charge sur les cœurs en utilisant une heuristique inspirée du problème de "bin-packing" de type "decreasing worst fit" pour un nombre limité de boîtes (les cœurs). Une approche

"worst fit" consiste à choisir la boîte qui a le plus d'espace libre lors du placement d'un objet. Pour notre problème, cette approche permet donc de lisser la charge allouée sur les différents cœurs.

2.3.2 Algorithme de partitionnement

Dans un premier temps, nous cherchons à respecter les contraintes d'allocation. Pour cela, nous regroupons donc les runnables avec des contraintes de co-localisation dans des "lots" et nous allouons les runnables ayant des contraintes de localisation. Ensuite, on place au fur et à mesure les lots de runnables sur le cœur le moins chargé en commençant par les groupes représentant le plus de charge afin d'équilibrer autant que possible à la fin avec les lots de runnables les moins importants en charge. La charge induite par un lot est donnée par le taux d'utilisation $\rho_{lot} = \sum_i \frac{C_i}{T_i}$ pour i désignant itérativement les runnables appartenant au lot. Cette heuristique est décrite dans l'algorithme 2.1.

Algorithme 2.1 Partitionnement d'un ensemble de runnables

en entrée : l'ensemble des runnables $\{\mathscr{R}_i\}$, le nombre de cœurs m
(1) Regrouper les runnables avec des contraintes de co-localisation dans des lots. Les runnables sans contraintes de co-localisation forment des lots de taille 1.
(2) Allouer les lots qui ont des contraintes de localisation sur le cœur correspondant.
(3) Trier les lots de runnables par ordre décroissant de taux d'utilisation $\rho_{lot} = \sum_i \frac{C_i}{T_i}$.
(4) En itérant sur les lots ainsi triés :
  (a) Trouver le cœur le moins chargé parmi les m cœurs,
  (b) Allouer ce lot sur ce cœur.

L'étape (1) s'exécute en $\mathscr{O}(n)$. L'étape (2) s'exécute aussi en $\mathscr{O}(n)$ mais les lots de runnables alloués pendant (2) n'auront pas besoin de passer à travers les étapes (3) et (4) qui sont plus complexes algorithmiquement. L'étape (3) s'exécute en $\mathscr{O}(n \cdot \log n)$. Enfin, l'étape (4) s'exécute en $\mathscr{O}(n \cdot m)$. En conclusion, l'algorithme 2.1 s'exécute en $\mathscr{O}(n(m + \log n))$ ce qui ne pose aucune difficulté en pratique [2].

Une première condition nécessaire d'ordonnançabilité est de vérifier que $m \geqslant \left\lceil \sum_{i=1}^{n} \frac{C_i}{T_i} \right\rceil$. En effet, il est nécessaire de s'assurer que le nombre de cœurs est suffisant pour exécuter la charge de calcul totale de l'ensemble des runnables.

Enfin, il est aisé d'adapter cette heuristique au cas où les cœurs ne seraient pas identiques car opérés à des fréquences différentes. Cela nécessiterait juste de connaître les valeurs $C_{i,k}$ correspondant au temps d'exécution du runnable i sur le

2. Quelques secondes de calcul pour des ensembles de plusieurs centaines de runnables sur un processeur dual-core à 2.8 GHz

cœur k pour le calcul des taux d'utilisation des lots et des cœurs (pour trouver le cœur le moins chargé).

2.4 Lissage de la charge sur une tâche séquenceur

L'étape suivante consiste donc à construire le séquencement des runnables alloués sur les cœurs. Dans un premier temps, nous faisons l'hypothèse qu'il n'y a pas de contrainte de précédence entre les runnables et qu'il n'y a qu'une seule tâche séquenceur par cœur. Pour des contraintes de précédence simples, il est de toute façon possible de concaténer les runnables dépendants et de les traiter dans l'algorithme comme un seul runnable avec un plus grand temps d'exécution. Nous aborderons en section 2.5 le cas où l'on considère plusieurs tâches par cœur.

Pour évaluer la qualité des tables de séquencement obtenues, nous utilisons deux critères. Le premier critère est d'avoir le plus petit pic de charge dans le cycle (*i.e.*, la plus petite charge maximale) car cela intervient directement pour valider la faisabilité de l'ordonnancement et permet de quantifier la facilité d'ajouter des fonctions supplémentaires. Le second critère consiste à regarder l'écart type de la distribution de charge dans les slots pour évaluer la qualité du lissage de la charge dans le cycle.

Dans la suite, on distingue deux cas : le cas harmonique et le cas non harmonique. Le cas dit harmonique correspondant au cas où les valeurs des périodes des runnables sont des multiples des périodes plus petites. Par exemple, l'ensemble de période {10, 20, 100, 200, 400} est harmonique tandis que {10, 20, 50, 100} ne l'est pas car 50 n'est pas un multiple de 20. La régularité caractéristique du cas harmonique permettra de donner des garanties de performances pour l'algorithme de choix des offsets.

2.4.1 Enoncé du problème

Nous cherchons à déterminer $\Omega(\{R\}_{p,k})$, la table d'ordonnancement d'une tâche séquenceur, en respectant les contraintes de périodicité des runnables et dans l'objectif de minimiser la charge maximum allouée parmi tous les slots.

Considérons un runnable $\mathscr{R}_i$ de période T_i, il existe $\frac{T_i}{T_{tic}}$ possibilités de choix pour l'offset initial de ce runnable. En effet, la seconde condition énoncée dans la section 2.2.4 impose de choisir un instant plus petit que la période et le modèle impose que cet offset initial soit un multiple de T_{tic} (à cause de l'utilisation d'une table d'ordonnancement avec des slots de taille T_{tic}). En conséquence, il existe $\prod_{i=1}^{n} \frac{T_i}{T_{tic}}$ possibilités de séquencements pour n runnables. Etant donné la complexité, nous ne connaissons pas d'algorithme optimal qui ne soit pas de complexité exponentielle. Prenons l'exemple de 50 runnables dont les périodes sont au moins le double de

T_{tic}, il faudrait évaluer au moins 2^{50} solutions distinctes.

2.4.2 Algorithme "Least Loaded" dans le cas harmonique

Une fois de plus, au regard de la complexité du problème nous recourons à des heuristiques. Ici, nous adaptons au problème d'ordonnancement des runnables l'algorithme "Least Loaded", abrégé LL, proposé par Grenier et al. dans [24] pour le problème de l'allocation des offsets pour des trames CAN.

L'intuition derrière l'heuristique est simple : à chaque étape, le runnable considéré est alloué dans le slot le moins chargé de la table d'ordonnancement comme décrit dans la trame de l"algorithme 2.2. La charge d'un slot est égale à la somme des C_i des runnables $\mathscr{R}_i$ déjà placés dans ce slot. La valeur de la charge d'un slot est mise à jour lorsque qu'un runnable est placé dans un slot en l'incrémentant de la valeur du WCET de ce runnable.

Algorithme 2.2 Allocation des runnables dans les slots : algorithme "Least Loaded"

en entrée : l'ensemble des runnables $\mathscr{R}_i$, T_i, T_{cycle}
(1) Ordonner les runnables $\mathscr{R}_i$ de sorte que $T_{tic} \leq T_1 \leq \cdots \leq T_n \leq T_{cycle}$
(2) Pour $i = 1 \cdots n$
  (a) Chercher le meilleur slot k le moins chargé dans les premiers $\frac{T_i}{T_{tic}}$ slots,
  (b) Placer $\mathscr{R}_i$ tous les $\frac{T_i}{T_{cycle}}$ slots en commençant par le slot k (cela revient à choisir $O_i = k \cdot T_{tic}$).

L'étape (1) s'exécute en $\mathscr{O}(n \cdot \log n)$. L'étape (2) itère n fois sur les étapes (2a) et (2b) s'exécutant respectivement en $\frac{T_i}{T_{tic}} \leq \frac{T_{cycle}}{T_{tic}}$ et $\frac{T_{cycle}}{T_i} \leq \frac{T_{cycle}}{T_{tic}}$. En conséquence, l'algorithme s'exécute en $\mathscr{O}(n(\log n + \max_i\{T_i\}/T_{tic} + T_{cycle}/\min_i\{T_i\})) \leq \mathscr{O}(n(\log n + 2 \cdot T_{cycle}/T_{tic}))$.

Dans la pratique, les conflits à l'étape (1) sont résolus en choisissant prioritairement le runnable avec le plus grand WCET et le meilleur slot à l'étape (2a) est défini comme étant le slot au centre de la plus grande série de slots consécutifs de charge la plus basse. Si cette dernière pratique n'a aucune influence sur les résultats théoriques de l'algorithme, elle a l'avantage de disperser les pics de charges, ce qui est important pour les concepteurs. La figure 2.3 montre le séquencement obtenu en appliquant l'heuristique LL ("Least Loaded") à l'ensemble de runnables $\mathscr{R}_i(T_i, C_i)$ suivants : $\mathscr{R}_1(10,2)$, $\mathscr{R}_2(10,1)$, $\mathscr{R}_3(20,3)$ et $\mathscr{R}_4(20,2)$. Pour $T_{tic} = 5$ et $T_{cycle} = 40$, cela donne une table d'ordonnancement avec 8 slots et séquencée comme représenté dans la figure 2.3.

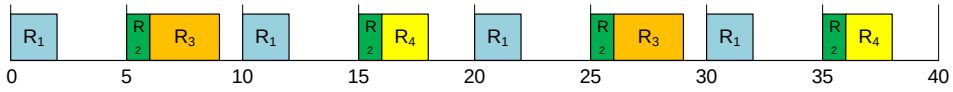

Figure 2.3 – Exemple de tâche séquenceur

Slot	1	2	3	4	5	6	7	8
Charge	2	4	2	3	2	4	2	3

Table 2.1 – Répartition de la charge dans la table d'ordonnancement correspondant à la figure 2.3

Borne supérieure sur le pic de charge maximal

Nous dérivons à présent une borne supérieure sur le pic de charge maximal obtenu après utilisation de l'algorithme "Least-Loaded" dans le cas d'un ensemble de runnables ayant des périodes harmoniques entre elles. Pour ce cas particulier, nous considérons également une condition d'ordonnançabilité plus stricte. Le résultat de l'algorithme est considéré comme acceptable si les runnables ont une charge inférieure à T_{tic}. On interdit donc le dépassement d'un slot sur le slot suivant. Cette restriction est pessimiste mais est une condition suffisante garantissant le respect des échéances (supérieure ou égale à T_{tic} par définition).

Dans un premier temps, nous montrons que les slots où $\mathscr{R}_i$ sera périodiquement placé, une fois son offset choisi, ont tous une charge égale.

Lemme 1. *Avant de placer $\mathscr{R}_i$ dans les slots, l'allocation des runnables déjà placés dans la table de séquencement se répète avec une période de $\frac{T_i}{T_{tic}}$.*

Preuve : Ce résultat est obtenu par récurrence. La propriété est vraie à l'état initial, au moment de placer $\mathscr{R}_0$ car tous les slots sont vides. Nous supposons à présent que l'allocation des runnables déjà placés soit périodique de période $\frac{T_i}{T_{tic}}$ au moment de placer $\mathscr{R}_i$. L'allocation sera toujours périodique après l'allocation de $\mathscr{R}_i$ car il sera placé tous les $\frac{T_i}{T_{tic}}$ slots. Comme les runnables sont classés par ordre croissant de période et que leurs périodes sont harmoniques, alors nous avons $T_{i+1} = k \cdot T_i$ avec $k \in \mathbb{N}^*$. L'allocation des runnables se répète donc aussi avec la période $k \cdot \frac{T_i}{T_{tic}} = \frac{T_{i+1}}{T_{tic}}$ avant de placer $\mathscr{R}_{i+1}$.■

Ce résultat justifie l'idée de ne regarder que les premiers $\frac{T_i}{T_{tic}}$ slots dans l'étape (2a) de l'algorithme 2.2.

Lemme 2. *Au moment de placer $\mathscr{R}_i$ dans les slots, le plus grande charge dans un slot qui puisse résulter de cette allocation est obtenue dans le cas d'un lissage parfait de la charge sur l'ensemble du cycle.*

Preuve : Ce résultat est intuitif. Comme l'ensemble des runnables déjà alloué induit une charge fixe dans le cycle, tout autre résultat qu'un lissage parfait comportera un slot moins chargé que dans le cas d'un équilibrage parfait de la charge. La plus grande charge obtenue après le placement de $\mathscr{R}_i$ sera donc plus basse car il sera placé dans un slot dont la charge était plus basse.■

En conséquence, le plus grand pic de charge que le placement d'un runnable puisse produire est obtenu dans le cas d'un lissage parfait de la charge avant son allocation. Nous définissons à présent le taux d'utilisation du cœur k : $\rho_k = \sum_{i \in \{\mathscr{R}\}_k} \frac{C_i}{T_i}$ où $\{\mathscr{R}\}_k$ désigne ici l'ensemble des runnables alloués sur le cœur k.

Théorème 1. *Sur le cœur* k*, la borne maximum* PL_k *sur la charge d'un slot est définie ainsi :*

$$\mathrm{PL}_k = \max_{i \in \{\mathscr{R}\}_k} \{\rho_k \cdot T_{tic} - \sum_{j=i}^{n_k} \frac{C_j}{T_j} \cdot T_{tic} + C_i\} \tag{2.1}$$

avec $n_k = \mathrm{card}\{\mathscr{R}\}_k$ *le cardinal de l'ensemble* $\{\mathscr{R}\}_k$ *(c'est à dire le nombre de runnables ordonnancée par la tâche séquenceur).*

Preuve : Au moment de placer $\mathscr{R}_i$, et dans le pire cas d'un équilibrage parfait, la charge de chaque slot est donnée par :

$$\sum_{runnables\ \ alloues} wcet \cdot \frac{nombre\ deslots\ ou\ le\ runnable\ est\ place}{nombre\ \ total\ \ deslots}, \ i.e$$

$$\sum_{j=1}^{i-1} C_j \cdot \frac{T_{cycle}}{T_j} \cdot \frac{T_{tic}}{T_{cycle}}$$

Après l'allocation de $\mathscr{R}_i$, la charge des slots où il est alloué est alors :

$$C_{i+} \sum_{j=1}^{i-1} C_j \cdot \frac{T_{tic}}{T_j}$$

De plus nous remarquons que : $\sum_{j \in \{\mathscr{R}\}_k}^{i-1} \frac{C_j}{T_j} = \rho_k - \sum_{j=i}^{n_k} \frac{C_j}{T_j}$. En conséquence, le pire pic de charge sur le cœur k induit par $\mathscr{R}_i$ est :

$$\mathrm{PL}_k^i = C_i + \rho_k \cdot T_{tic} - \sum_{j=i}^{n_k} \frac{C_j}{T_j} \cdot T_{tic} \tag{2.2}$$

Le résultat (2.1) est obtenu en prenant le maximum possible sur l'ensemble des runnables. ■

Afin de réduire la complexité, nous utilisons une condition de faisabilité plus contraignante pour les preuves suivantes. La solution trouvée est considérée faisable si la charge dans les slots est plus petite que la longueur des slots, ce qui est plus contraignant que le respect des échéances sur requête. En effet, si le pire pic induit est inférieur à T_{tic} alors la dernière condition d'ordonnançabilité de 2.2.4 est respectée. D'où le corollaire suivant :

Corollaire 1. *Le théorème 1 mène à la condition suffisante d'ordonnançabilité suivante :*

$$\rho_k \leq 1 - \frac{C_{max}}{T_{tic}} + \frac{C_{min}}{T_{max}} \tag{2.3}$$

Avec $C_{max} = \max_{i \in \{\mathscr{R}\}_k}\{C_i\}$, $C_{min} = \min_{i \in \{\mathscr{R}\}_k}\{C_i\}$ *et* $T_{max} = \max_{i \in \mathscr{R}_k}\{T_i\}$.

Preuve : A partir de (2.2), $\forall i, C_i \leq C_{max}$ et $\forall i, \sum_{j=i}^{n_k} \frac{C_j}{T_j} \geq \frac{C_{min}}{T_{max}}$ donne :

$$\forall i, \mathrm{PL}_k^i \leq C_{max} + \rho_k \cdot T_{tic} - \frac{C_{min}}{T_{max}} \cdot T_{tic}$$

De plus, si le pic de charge maximal PL_k^i est plus petit que la longueur d'un slot T_{tic} alors les échéances seront toutes respectées. Il s'agit d'une condition suffisante et non nécessaire. En effet, T_{tic} est inférieur aux échéances de tous les runnables (par définition, il divise leurs périodes qui sont égales à leurs échéances). Ainsi, $\mathrm{PL}_k^i \leq T_{tic}$ donne le résultat voulu.■

Cette borne est atteignable pour un ensemble de $n \cdot k$ runnables identiques de période égale à $k \cdot T_{tic}$ et de charge C et un dernier runnable $\mathscr{R}_{n \cdot k+1}$ de période T_{max} et de charge C. Pour cet ensemble, on a $C_{max} = C_{min} = C$. L'allocation des $n \cdot k$ premiers runnables résulte en un lissage parfait de la charge de moyenne $\rho_k \cdot T_{tic} - C \cdot T_{tic}/T_{max}$. Enfin, l'allocation du dernier runnable induit la charge $\rho_k \cdot T_{tic} - C \cdot T_{tic}/T_{max} + C_{max}$ dans les slots où il sera placé.

Condition suffisante de l'allocation d'un niveau de charge sur un cœur

Nous déduisons ici une borne sur la capacité des cœurs que l'heuristique garantit de pouvoir utiliser pour un ensemble de runnables avec des périodes harmoniques. Cette borne est appelée "borne harmonique d'ordonnancement".

Définition La "borne harmonique d'ordonnancement" ρ_h est égale à $(1 - \frac{C_{max}}{T_{tic}})$ de la puissance de calcul de chaque cœur.

Théorème 2. $\rho_k < \rho_h$ *est une condition suffisante d'ordonnancement pour le cœur k.*

Preuve : En raisonnant de manière similaire au corollaire 1, le pire cas de pic de charge est obtenu en plaçant le dernier runnable $\mathscr{R}(C_{max}, T_{max} = T_{tic})$ dans le cas d'un lissage parfait de la charge déjà allouée. Dans ce cas critique, le système est encore ordonnançable si la charge allouée dans chaque slot est inférieure ou égale à $T_{tic} - C_{max}$. En d'autres termes, quand le système ne devient plus ordonnançable, chaque slot a une charge supérieure ou égale à $T_{tic} - C_{max}$. En conséquence, il est

garanti de pouvoir utiliser au moins $(1 - \frac{C_{max}}{T_{tic}})$ de la puissance de calcul du cœur en utilisant notre algorithme.■

Par exemple, avec $T_{tic} = 5\,ms$ et $C_{max} = 300\,\mu s$, il est donc garanti que l'on puisse utiliser le cœur à au moins 94%. En pratique, C_{max} étant relativement petit, cette borne est donc assez haute. De plus, cette borne repose sur la contrainte forte exigeant que les runnables alloués dans un slot doivent avoir fini leur exécution avant la fin de celui-ci.

Dans la pratique, les échéances des runnables sont égales à leur période et il est autorisé de continuer son exécution sur les slots suivants l'activation d'un runnable : il est donc possible d'utiliser encore plus la capacité des cœurs. Nous rappelons de plus que cette borne est pessimiste par rapport au cas où l'on considère que les échéances des runnables sont égales à leur période et où il est possible de surcharger un slot (comme nous allons le voir ensuite).

Corollaire 2. *Nous donnons à présent une borne supérieure sur le nombre minimum de cœurs nécessaires pour ordonnancer un ensemble de runnables. Soit* $\mathrm{P} = \sum_i \frac{C_i}{T_i}$ *la charge totale induite par l'ensemble des runnables et* m_{min} *le nombre minimum de cœurs nécessaires à l'ordonnancer, alors le théorème 2 donne :*

$$m_{min} \leq \left\lceil \frac{\mathrm{P}}{1 - C_{max}/T_{tic}} \right\rceil$$

2.4.3 Algorithme "Lowest-Peak" pour le cas non harmonique

En pratique, les ensembles de runnables utilisés n'ont pas toujours des périodes strictement harmoniques. En conséquence, les lemmes 1 et 2 ainsi que les résultats théoriques en résultant ne peuvent plus être utilisés pour fournir des bornes sur la charge dans les slots. En particulier, placer un runnable dans le slot le moins chargé peut provoquer des pics de charges quand on cherche à le placer dans les slots suivants, du fait de sa périodicité. Prenons l'ensemble de runnables suivants par exemple : $\mathscr{R}_1(10,2)$, $\mathscr{R}_2(20,3)$, $\mathscr{R}_3(20,1)$ et $\mathscr{R}_4(50,2)$ avec $T_{tic} = 5$ et $T_{cycle} = 100$. La figure 2.4 montre la tâche séquenceur avant l'allocation de $\mathscr{R}_4$.

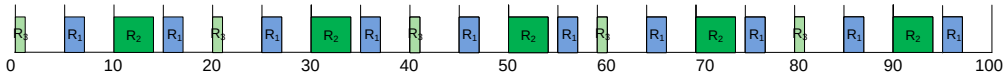

Figure 2.4 – tâche séquenceur avant l'allocation de $\mathscr{R}_4$

Au moment de placer $\mathscr{R}_4$, choisir un des slots les moins chargés donnerait un séquencement invalide car il serait ultérieurement placé dans un des slots avec la charge la plus grande du fait de sa périodicité. Par exemple, placer la première instance de $\mathscr{R}_4$ dans le slot 1 aurait pour conséquence de devoir placer la seconde

Slot	1	2	3	4	5	6	7	8	9	10	11	12	...
Charge	1	2	4	2	1	2	4	2	1	2	4	2	...

Table 2.2 – Répartition de la charge de la table d'ordonnancement pour la figure 2.4

instance de $\mathscr{R}_4$ dans le slot 11 et résulterait en un ordonnancement non faisable. Cependant, placer $\mathscr{R}_4$ dans un slot pair n'est pas risqué.

Algorithme 2.3 Heuristique "Lowest Peak" pour les ensembles de runnables avec des périodes non harmoniques

en entrée : l'ensemble des runnables $\mathscr{R}_i$, T_i, T_{cycle}
(1) Ordonner les runnables $\mathscr{R}_i$ de sorte que $T_{tic} \leq T_1 \leq \cdots \leq T_n \leq T_{cycle}$
(2) $T_{window} = T_{tic}$
(3) Pour $i = 1 \cdots n$
(a) $T_{window} = \text{PPCM}(T_{window}, T_i)$,
(b) Dans les premiers T_i/T_{tic} slots, chercher le slot tel que le plus grand pic de charge induit par le placement périodique de $\mathscr{R}_i$ dans les premiers T_{window}/T_{tic} slots soit le plus petit,
(c) Placer $\mathscr{R}_i$ tous les $\frac{T_i}{T_{cycle}}$ slots en commençant par ce slot.

Dans le but de prendre en charge les ensembles de runnables avec des périodes non harmoniques, il faut donc regarder un plus grand ensemble de slots (car le lemme 1 ne tient plus). Dans la suite, T_{window} est égal au PPCM des périodes des runnables préalablement traités par l'algorithme. Ainsi, au lieu de chercher le slot le moins chargé dans les premiers T_i/T_{tic} slots, nous cherchons le slot induisant le plus petit pic dans les premiers T_{window}/T_{tic} slots après lesquels l'ordonnancement se répétera. Dans cet évolution baptisée "Lowest Peak" (ou LP) , nous changeons essentiellement le critère de placement. Le nombre de cas à étudier est plus important que dans le premier algorithme mais reste néanmoins dans le même ordre de grandeur.

L'étape (1) de l'algorithme 2.3 s'exécute en $\mathcal{O}(n \cdot \log n)$. L'étape (3a) s'exécute en $\mathcal{O}(\log T_{cycle})$. Enfin, les étapes (3b) et (3c) s'exécutent respectivement en $\mathcal{O}(n \cdot T_{window}/T_{tic}) \leq \mathcal{O}(n \cdot T_{cycle}/T_{tic})$ et $\mathcal{O}(n \cdot T_{cycle}/T_i) \leq \mathcal{O}(n \cdot T_{cycle}/T_{tic})$. En conséquence, l'algorithme complet s'exécute donc en $\mathcal{O}(n \cdot (\log n + 2 \cdot T_{cycle}/T_{tic} + \log T_{cycle}))$.

2.4.4 Compléments sur les algorithmes de base

Exécution d'un runnable sur plusieurs slots consécutifs

En pratique, chaque slot a une capacité limitée. La charge maximum exécutable pendant la durée d'un slot est naturellement la durée d'un slot (T_{tic}). Si la charge allouée dans un slot dépasse sa capacité, cela va retarder l'exécution des runnables dans le slot suivant. Ainsi lors de l'étape (3b), si le placement d'un runnable fait dépasser la capacité d'un slot, la charge supplémentaire est reportée sur les slots suivants tant que ceux-ci atteignent également leur capacité maximale. La charge induite par les runnables doit être inférieure à la capacité du cœur sinon le système ne serait pas ordonnançable. En conséquence, le report de la charge finira par s'arrêter. Dans le cas où le placement d'un runnable fait nécessairement dépasser la capacité d'un des slot où il sera alloué, le plus petit pic de charge induit par le placement des runnables sera à chercher dans les slots où la charge supplémentaire a été reportée.

"Lowest Peak - sigma" pour les runnables avec de grands WCET

Les algorithmes présentés ici permettent de construire l'ordonnancement des runnables avec des périodes arbitraires et d'éventuelles contraintes de localisation et de co-localisation. Cependant, comme nous le montrerons dans les expérimentations à venir, les algorithmes de choix d'offset donnent des résultats moins satisfaisants dans le cas où les ensembles de runnables comportent un petit nombre de runnables avec de grands WCET par rapport aux autres runnables.

En pratique, les runnables avec de grands WCET sont généralement cadencés avec de plus grandes périodes. En conséquence, ceux-ci sont souvent traités en dernier lors de l'emploi des heuristiques présentés ce qui crée de grands pics de charge. Dans le but de contrer ce phénomène, nous avons modifié les algorithmes afin de placer en premier ces runnables de façon à lisser la charge avec le reste des runnables et diminuer ainsi les pics de charge.

Nous définissons un WCET seuil $C_{critic} = \mu + k \cdot \sigma$ avec μ et σ respectivement la moyenne et l'écart type de la distribution des $\{C_i\}$ et k un entier. Les runnables $\mathscr{R}_i$ tels que $C_i \geq C_{critic}$ sont donc traités en premier dans l'algorithme 2.3. Cette nouvelle version de l'heuristique est baptisée "Lowest Peak - sigma" ou $\mathrm{LP}_{k\sigma}$. Dans les expérimentations suivantes nous utiliserons $k = 1$. Nous avons également essayé d'ordonner la totalité des runnables par ordre décroissant de WCET mais cette approche s'est révélée bien moins efficace dans la pratique, au niveau du lissage.

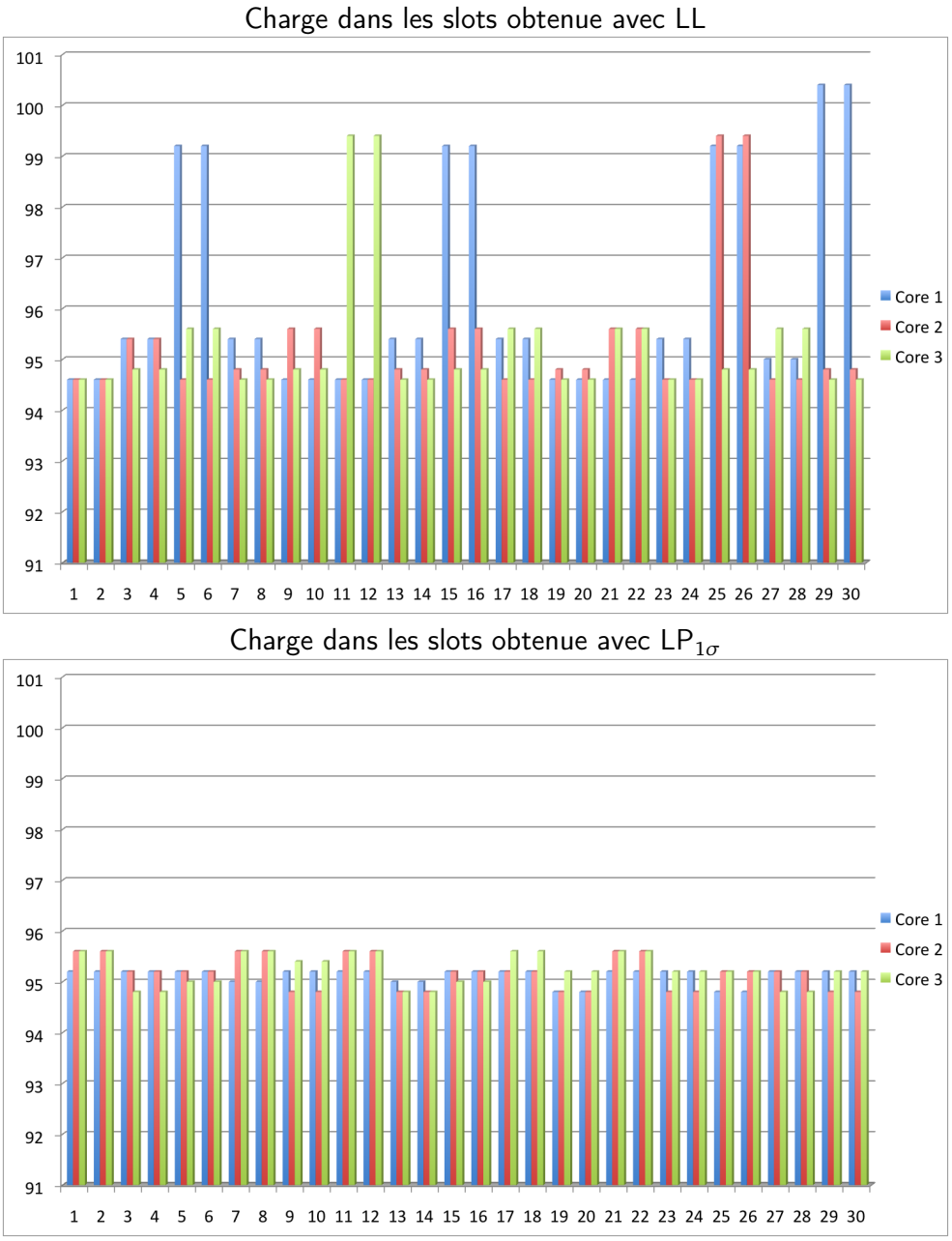

Figure 2.5 – Ensemble de runnables ordonnancés sur 3 cœurs pour une charge proche de 94% avec respectivement LL et LP$_{1\sigma}$ à haut et en bas. Ces graphiques représentent la charge (en ordonnée) des slots (en abscisse) des tâches séquenceurs de chacun des cœurs. Pour des questions de lisibilité, nous ne présentons ici que les 30 premiers slots. Ces graphiques ont été obtenus avec NETCAR-ECU.

2.4.5 Expérimentations

Dans cette partie, nous évaluons les performances des algorithmes d'ordonnancement présentés sur des exemples caractéristiques des systèmes embarqués dans les véhicules. Les algorithmes LL, LP, et $\mathrm{LP}_{k\sigma}$ décrits respectivement en 2.4.2, 2.4.3 et 2.4.4 ont été implémentés comme plug-in de NETCAR-ECU [62]. Les caractéristiques des runnables sont tirés aléatoirement sur la base d'une distribution réelle d'un BSI [3]. Nous cherchons à mettre en évidence la capacité des heuristiques à harmoniser la charge sur la longueur du cycle d'ordonnancement ainsi qu'à supporter de très hauts niveaux de charge.

Equilibrage de la charge

Pour faire ces expérimentations, nous donnons aux runnables le même niveau de priorité afin de n'avoir qu'une seule tâche séquenceur par cœur. La distribution d'origine comporte environ 200 runnables dont les périodes sont presque harmoniques (seulement 5% des runnables n'ont pas une période harmonique avec les autres). La durée des slots est fixée à 5ms pour l'ensemble des expérimentations Nous appliquons directement les algorithmes de lissage une fois le partitionnement obtenu.

L'ensemble de runnables de référence possède des périodes presque harmoniques et il suffirait de retirer quelques runnables pour se retrouver dans le cas harmonique. Pour cette raison, nous dérivons une distribution de période plus difficile pour les algorithmes car plus éloignée du cas harmonique en ajoutant des périodes supplémentaires et en modifiant la distribution de manière à s'écarter encore plus du cas harmonique (en augmentant la chance de tomber sur les périodes qui n'étaient pas utilisées dans le cas précédent). L'ensemble des périodes utilisées est le suivant :

$$\{10, 20, 25, 40, 50, 100, 125, 200, 250, 500, 1000\text{ms}\}$$

Les WCETs varient entre 10μs et 300 μs. Les périodes et les WCETs sont générés sur la base de probabilités dérivées de la distribution observée sur un calculateur existant. Nous continuons à générer des runnables jusqu'à atteindre une charge légèrement inférieure à 94% afin de s'assurer de la faisabilité. Le graphique en haut de la figure 2.5 montre un exemple de distribution de charge sur un cycle obtenu sur 3 cœurs avec l'algorithme LL et le graphique du bas donne le résultat obtenu en utilisant l'algorithme $\mathrm{LP}_{1\sigma}$ présenté en 2.4.4.

3. Boîtier de Servitude Intelligent. Il s'agit du calculateur qui gère le tableau de bord et qui fait l'interface entre les réseaux corresponsdant aux différents domaines fonctionnels d'un véhicule (habitacle, télématique, sous-capot... etc)

Performance et robustesse des algorithmes

A présent, nous essayons d'étudier dans quelle mesure la borne harmonique d'ordonnancement tient dans le cas non harmonique. Précisément, nous évaluons le taux de réussite des algorithmes pour des ensembles de runnables avec des périodes non-harmoniques dans des conditions garantissant la faisabilité dans le cas harmonique (d'après le théorème 2). À cette fin, nous utilisons ici un critère plus strict pour le succès de l'ordonnancement. Nous considérons que l'ordonnancement des runnables est réussi si la charge de chaque slot est inférieure à la longueur d'un slot T_{tic}. Les distributions de temps de réponse dans les cas où le WCET maximum est égal à 150 μs ou 900 μs sont obtenues respectivement en divisant par 2 et en multipliant la distribution initiale.

Comme nous pouvons le voir dans la table 2.3, quand la charge est proche de la borne harmonique d'ordonnancement, les heuristiques proposées restent très efficaces, et en particulier $LP_{1\sigma}$ qui a été capable d'ordonnancer avec succès les 1000 configurations générées aléatoirement que nous lui avons soumises. Ceci suggère que cette borne est un premier critère de dimensionnement envisageable, même dans le cas non harmonique.

WCET max (μs)	150	300	900
Borne harmonique	97%	94%	82%
Taux de réussite de LL dans le cas non-harmonique	96%	96%	92%
Taux de réussite de $LP_{1\sigma}$dans le cas non-harmonique	100%	100%	100%

Table 2.3 – Performance des algorithmes dans le cas non-harmonique pour des charges proches de la borne harmonique d'ordonnancement. Statistiques collectées sur 1000 configurations générées pour chaque cas.

La table 2.4 donne les résultats obtenus pour des charges plus importantes, c'est-à-dire plus hautes que la borne harmonique d'ordonnancement.

Charge CPU générée	95%	97%	95%	97%
Borne harmonique	94% (max WCET=300μs)		82% (max WCET=900μs)	
Taux de réussite de LL	64%	18%	12%	1%
Taux de réussite de LP	94%	94%	30%	5%
Taux de réussite de $LP_{1\sigma}$	100%	100%	97%	76%

Table 2.4 – Performance des algorithmes d'ordonnancement dans le cas non harmonique pour des charges plus grandes que la borne harmonique d'ordonnancement. Statistiques collectées pour 1000 configurations générées pour chaque cas.

De manière prévisible, plus on s'éloigne de la borne harmonique, plus il est

difficile de trouver un ordonnancement valide. La seconde remarque est que l'heuristique $LP_{1\sigma}$ donne de loin les meilleurs résultats, en particulier dans le cas où les pires temps d'exécutions deviennent grands (avec le WCET maximum à 900μs).

2.5 Lissage de charge avec plusieurs tâches séquenceurs par cœur

A présent, nous abordons le cas général où plusieurs tâches sont ordonnancées sur un même cœur. Tout d'abord, nous considérons le cas général où il peut y avoir plusieurs tâches séquenceurs sur un même cœur (correspondant à différents niveaux de priorité). Comme AutoSar OS ne fournit des mécanismes de protection mémoire qu'au niveau des tâches, ce cas arrive par exemple si de la protection mémoire est requise entre certains runnables. En effet, les runnables sont transparents pour le système d'exploitation, car ils sont séquencés au sein d'une tâche OS. Dans ce cas, nous faisons l'hypothèse que les différentes tâches OS sont ordonnancées selon un ordonnanceur à priorités fixes comme cela est fortement probable dans les calculateurs embarqués dans l'automobile. Enfin, en sus des tâches séquenceurs qui sont construites par les algorithmes présentés dans ce chapitre, le processeur sera en charge d'exécuter d'autres tâches dont nous connaissons les caractéristiques mais que nous ne pouvons pas modifier.

Dans cette partie, nous distinguons deux cas : le cas de tâches synchrones et le cas de tâches asynchrones. Dans le premier cas, les offsets entre les tâches sont connus (mais non nécessairement nuls) et le système est cadencé par une horloge unique (les tâches sont ordonnancées selon une base de temps commune). Dans le second cas, les tâches sont cadencées par des horloges différentes. Ce dernier cas survient, par exemple, pour les calculateurs utilisés pour le contrôle moteur où une partie des runnables est cadencéé par l'horloge du microcontrôleur tandis que le reste des runnables est cadencé sur le rythme du régime moteur (du point mort haut) qui varie dans le temps.

Nous étudions tout d'abord en détail le cas de tâches synchrones et aborderons succinctement le cas asynchrone à la fin de cette partie. Pour les mêmes raisons que dans la partie précédente, l'approche se décompose successivement en une étape de partitionnement des tâches sur les différents cœurs puis en une étape de construction des tâches séquenceurs, faite sur chacun des cœurs, indépendamment les uns des autres. Une dernière étape consiste à vérifier si les contraintes d'échéances sont respectées.

2.5.1 Modèle des tâches

Tout d'abord, nous distinguons deux types de tâches : les "tâches séquenceurs," qui servent à ordonnancer les runnables, et les "tâches fixes" qui nous sont données

et doivent être ordonnancées sur le même processeur. Les tâches ont chacune un niveau de priorité et l'ordonnancement sur les cœurs est réalisé selon une politique FPP (ordonnancement préemptif à priorités fixes). De plus, les tâches peuvent démarrer après un offset initial qui leur est propre. Ici, nous considérons deux types de tâche fixes : les tâches périodiques et les tâches multiframes.

Tâches périodiques Les tâches périodiques sont décrites de manière classique et donc de manière similaire à un runnable : une tâche i sera ainsi caractérisée par sa période T_i, son pire temps d'exécution C_i, son offset O_i et son niveau de priorité P_i. En pratique, elles sont similaires à des runnables avec des temps d'exécution plus longs et ordonnancées au niveau OS.

Tâches multiframes Les tâches multiframes suivent le modèle introduit par Mok et Chen [44] et sont des tâches périodiques dont le temps d'exécution varie d'une instance sur l'autre. Elles peuvent servir à décrire les tâches séquenceurs que l'on ne peut pas modifier ou alors la tâche de communication dans les cas où l'on peut estimer ses temps d'exécution successifs à partir de la connaissance de la messagerie CAN (généralement déjà définie à cet étape du développement de l'architecture électronique du véhicule). La tâche multiframe i sera caractérisée par un offset O_i, son niveau de priorité P_i, par la période interframe (entre deux frames successives) t_i, un vecteur décrivant les temps d'exécutifs des différentes frames consécutives $C_i = \{C_{i1}, C_{i2}, ..., C_{ik}\}$ pour k frames. Les temps d'exécution des instances consécutives sont prises dans ce vecteur de manière cyclique. Ainsi, l'instance $k+1$ de la tâche aura un temps d'exécution égal à C_{i1}. La période de la tâche est alors $T_i = k.t_i$ pour k frames correspondant à la macro-période T_{cycle} utilisée pour les tâches séquenceurs et qui est la période de la tâche utile pour calculer la période d'étude nécessaire à décrire l'ordonnancement d'un cœur.

Il est important de remarquer qu'il est très facile de décrire une tâche séquenceur par une tâche multiframe en transformant chaque slot en une frame dont le temps d'exécution est la somme des temps d'exécution des runnables contenus dans le slot. Cela peut être utile dans le cadre d'un développement incrémental à partir d'un processeur existant. Dans l'optique d'ajouter des runnables supplémentaires sur un calculateur sans modifier l'ordonnancement des runnables existants, il est possible de considérer les tâches séquenceurs préalablement construites comme des tâches multi-frames et d'ajouter une nouvelle tâche séquenceur pour les nouveaux runnables.

Tâches séquenceurs Par opposition aux "tâches fixes", on considère que les tâches séquenceurs ont un offset nul. Comme les algorithmes construisent les tâches séquenceurs en fixant les offsets des runnables, et sans contrainte de précédence au

niveau des runnables, il est inutile de choisir un offset initial différent de 0 pour les tâches séquenceurs. Enfin, nous faisons l'hypothèse que la longueur des slots T_{tic} est la même pour les tâches séquenceurs et qu'il est diviseur des périodes de tous les runnables, ainsi que des périodes de toutes les tâches périodiques et enfin des périodes interframes de toutes les tâches multi-frames.

Sur chaque cœur, nous supposons que les offsets de tous les runnables et "tâches fixes" sont choisis comme des multiples de T_{tic}. En conséquence, dans le cas synchrone, les slots des tâches séquenceurs commencent simultanément et coïncident avec les temps d'arrivée des différentes "tâches fixes".

2.5.2 Partitionnement des runnables

Comme dans la partie précédente et pour les mêmes raisons, on procède en deux étapes. Tout d'abord, on répartit les tâches sur les différents cœurs avant de considérer indépendamment chacun des cœurs afin de construire les tâches séquenceurs en choisissant les offsets initiaux des runnables. En pratique, les tâches fixes seront probablement déjà allouées sur des cœurs. Dans le cas contraire, l'algorithme de partitionnement des runnables 2.1 est aisément généralisable pour intégrer les tâches périodiques et multiframes. A l'étape (3) de l'algorithme, le taux d'utilisation d'une tâche périodique i est alors $\rho_i = C_i/T_i$ et le taux d'une tâche multiframe i est logiquement donnée par $\rho_i = \sum_j C_{ij}/T_i$. Dans le cas général, les tâches périodiques et les tâches multiframes devraient être placées relativement tôt lors de la mise en oeuvre de l'algorithme car elles induisent plus de charge qu'un runnable sans contrainte de colocalisation, qui a typiquement un temps d'exécution sensiblement plus faible.

2.5.3 Construction des tâches séquenceurs

Une fois les runnables répartis sur les différents cœurs, il convient donc de construire les tâches séquenceurs. Pour cela, on utilisera une approche reposant sur les algorithmes de lissage de charge présentés dans la partie précédente. Cependant, dans le cas synchrone, comme les offsets entre les différentes tâches sont connus et constants, il est possible de construire une tâche séquenceur en tenant compte des autres tâches ordonnancées sur le même cœur dont on a connaissance.

Période d'étude

Dans le cadre que nous avons choisi, le système est périodique. En effet, on peut considérer que les slots des tâches séquenceurs, tout comme les frames des tâches multiframes sont chacune des tâches périodiques de période égale à l'hyper-période T_{cycle} de la tâche dont ils dépendent. Ainsi, s'il est faisable, l'ordonnancement résultant sera périodique. En revanche, comme les tâches peuvent être à départ

différé, l'ordonnancement ne sera périodique qu'à partir d'un certain temps [23]. On distingue ainsi deux phases : une phase initiale pendant laquelle les tâches sont successivement déclenchées et une phase périodique qui commencera une fois que les effets de bord dûs au démarrage se seront résorbés. Cependant, comme l'ordonnancement est préemptif et à priorité fixe, le temps de réponse se produira nécessairement une fois cette phase périodique atteinte. En effet, le temps de réponse d'un runnable ou d'une tâche dépend uniquement de la quantité de travail plus prioritaire à exécuter sur le cœur au moment où une instance de cette tâche arrive. Le pire cas se produira donc une fois la phase initiale passée où il y a au plus autant de charge plus prioritaire. Ainsi, il est suffisant de considérer uniquement ce qu'il se passe une fois la phase périodique atteinte. Nous définissons donc une tâche séquenceur virtuelle composée de slots de longueur T_{tic} sur un intervalle de temps T_{etude} égal au $PPCM$ des périodes des tâches ordonnancées sur le cœur (en utilisant T_{cycle} pour les tâches séquenceur et les tâches multiframes). Cette tâche permet de décrire l'ordonnancement une fois le régime permanent atteint. Au fur et à mesure de la construction des tâches séquenceurs, les runnables et les tâches sont alloués dans cette tâche virtuelle. Cela permet donc de tracer la charge déjà allouée sur l'ensemble d'une période du système lors du choix de l'offset des runnables lors de l'application d'un algorithme de lissage de la charge.

Stratégies

En fonction des priorités de conception, plusieurs approches peuvent être empruntées pour construire les tâches séquenceurs sur un cœur. En effet, comme nous sommes en connaissance de la charge induite par les "tâches fixes", nous envisageons plusieurs moyens d'utiliser ces informations lors du choix des offsets des runnables.

Stratégie prioritaire Tout d'abord, l'approche la plus intuitive est de construire le séquencement des runnables en procédant itérativement par niveau de priorité. En commençant par les tâches les plus prioritaires, on remplit les slots de la tâche séquenceur virtuelle. S'il s'agit d'une "tâche fixe", la charge correspondante est allouée dans les slots de la tâche virtuelle sans oublier de reporter la charge sur les slots suivants dans le cas où la capacité d'un slot est atteinte. S'il s'agit d'une tâche séquenceur, on utilise LL, LP ou $\text{LP}_{k\sigma}$ pour choisir les offsets des runnables en utilisant la tâche virtuelle pour connaître la charge plus prioritaire allouée dans les slots. Avec cette approche, on tient uniquement compte des tâches plus prioritaires. De plus, les runnables les plus prioritaires sont placés en premier et auront donc des temps de réponse faibles. En revanche, les tâches les moins prioritaires subissent les choix faits pour les tâches plus prioritaires. Ainsi, si des tâches fixes ou des runnables de faible priorité ont des pire temps d'exécution importants, cela risque de créer des pics de charges. Cela est d'autant plus problématique pour le cas des tâches fixes

car leur placement est déterminé à l'avance. Ainsi, contrairement aux runnables, il n'est pas possible d'essayer de les placer de manière à limiter les problèmes de surcharge. On retrouve le problème qui a mené à proposer $\mathrm{LP}_{k\sigma}$. Cette approche est appelée "stratégie prioritaire".

Stratégie mixte Une stratégie alternative consiste à prendre avantage de la connaissance des tâches fixes moins prioritaires pour diminuer leur temps de réponse. Comme leur ordonnancement est connu à l'avance, on cherchera à éviter autant que possible de synchroniser les arrivées des runnables plus prioritaires afin de limiter les pics de charge et diminuer les temps de réponse des tâches moins prioritaires. Pour cela, on commence par allouer dans la tâche séquenceur virtuelle la charge induite par toutes les tâches fixes, quel que soit leur niveau de priorité. Ensuite seulement, nous construisons itérativement, une à une et par ordre de priorité, les tâches séquenceurs en utilisant des algorithmes de lissage de charge pour obtenir les offsets des runnables. En conséquence, les temps de réponses des tâches fixes moins prioritaires que des tâches séquenceurs construites par nos soins seront plus faibles. Cependant, le problème des runnables avec des grands WCET se pose toujours dans ce cas pour les runnables des plus faibles niveaux de priorité. Cette approche est appelée "stratégie mixte".

Stratégie globale Une dernière stratégie consiste à lisser la charge au niveau global, sans tenir compte des niveaux de priorité. Cela permet de limiter les pics induits par les runnables avec des grands WCET et un niveau de priorité faible. Cela a aussi pour conséquence de réduire les temps de réponses des runnables les moins prioritaires en faisant de petits sacrifices sur les temps de réponse des runnables de niveau de priorité intermédiaire. Comme pour la stratégie mixte, on commence par allouer dans la tâche séquenceur virtuelle. Ensuite, on applique LL, LP ou $\mathrm{LP}_{k\sigma}$ à l'ensemble des runnables, indépendamment de leur niveau de priorité pour calculer leurs offsets. Contrairement aux deux stratégies précédentes, si on considère deux runnables $\mathscr{R}_i$ et $\mathscr{R}_j$ tels que $T_j < T_i$, alors $\mathscr{R}_j$ sera alloué avant $\mathscr{R}_i$ même si $\mathscr{R}_i$ est plus prioritaire que $\mathscr{R}_j$. En conséquence, une fois tous les runnables allouées, la charge vue au niveau OS sera généralement mieux lissée par rapport au résultat des autres stratégies.

Ordre des runnables

Au sein d'un même niveau de priorité, l'ordonnancement des runnable respecte leur ordre d'arrivée. Dans le cas où une tâche séquenceur peut se faire préempter par une tâche plus prioritaire, l'ordre des runnables est important pour la vérification des échéances. En effet, comme ils sont exécutés séquentiellement, les premiers runnables d'un slot ont moins de chance de se faire interrompre pendant

leur exécution. Pour tenir compte de cela, l'ordre des runnables dans les slots est fixé en fonction de leur échéance (donc leur période). Ainsi, les runnables avec les échéances les plus proches (les périodes les plus courtes), sont placés en premier.

Afin d'avoir une vision fidèle de l'ordonnancement, on reconstruit fidélement la séquence des instances des tâches, frame et runnables de manière à respecter les règles d'ordonnancement. Pour cela, on procède itérativement par niveau de priorité à placer les runnables dans les slots en respectant les offsets calculés et leur période en commençant par la tâche la plus prioritaire. Les runnables d'un même niveau de priorité arrivant dans un même slot sont classés en plaçant en premier ceux dont l'échéance est la plus proche. Si un slot est rempli, le reste du temps à exécuter est reporté dans les slots suivants tant que nécessaire. Une fois ce travail réalisé, la tâche séquenceur virtuelle décrit fidèlement l'ordonnancement obtenu et son étude permet de vérifier le respect des contraintes d'échéances.

2.5.4 Vérification des contraintes de temps - cas périodique

Dans le cadre du système étudié, la vérification du respect des contraintes de temps sur un cœur consiste à vérifier que toutes les échéances sont vérifiées. Ce travail peut-être fait indépendamment sur chacun des cœurs. En étudiant la tâche séquenceur virtuelle du cœur étudié, la vérification des échéances est relativement aisée. Comme expliqué précédement, les cas les plus défavorables pour un ordonnancement FPP se dérouleront une fois le régime périodique établi. Ainsi, si toutes les échéances sont respectées en étudiant l'ordonnancement décrit par la tâche séquenceur virtuelle, alors la solution calculée est valide.

Une première vérification rapide à faire est de regarder si jamais tous les slots ont une charge inférieure à leur durée. Dans ce cas, nous sommes garantis que les échéances sont respectées car elles sont toutes supérieures ou égales à la durée d'un slot. Il s'agit d'une condition suffisante mais non nécessaire.

Pour cette même raison, les endroits où des échéances peuvent être manquées sont les séquences de slots consécutifs qui ont été entirèrement remplis ainsi que le slot directement consécutif. Une fois qu'un slot présente un temps creux, cela signifie qu'il n'y a plus de de charge de calcul en attente. Une fois ces sections identifiées, il est alors nécessaire de vérifier que tous les éléments contenus dans ces sections respectent leur échéance.

Pour procéder plus rapidement, il peut-être suffisant dans un premier temps de travailler par niveau de priorité. Dans ce cas, on filtre les tâches moins prioritaires et on cherche la date du premier temps creux. Si celui-ci est plus proche que la plus petite des échéances des éléments contenus dans le slot, alors aucune échéance ne sera manquée pour ce niveau de priorité.

Prise en compte de la gigue

Dans le modèle que nous considérons, nous avons ignoré le phénomène de gigue. Ainsi, les algorithmes choisissant les offsets ne tiennent pas compte de la gigue. Cependant, il est tout de même possible d'utiliser les algorithmes présentés car si le lissage de la charge est bon, cela apporte de la robustesse vis-à-vis du phénomène de gigue même si la solution obtenue n'est pas optimale. Si la gigue est faible, les algorithmes devraient être capables de donner une solution valide dans la plupart des cas. La méthode publié dans [64] permet alors de calculer les temps de réponses. Pour cela, on décrit chacun des runnables comme une tâche périodique dans leur modèle. De plus les frames des tâches multiframes sont chacune transformées en une tâche périodique de période égale à l'hyper-période de la tâche multiframe à laquelle elles appartiennent. En cas de forte gigue, il convient d'opérer comme dans le cas asynchrone étudié ci-après.

2.5.5 Cas asynchrone

L'approche précédente ne peut malheureusement pas être appliquée à des taches asynchrones car leurs offsets et leurs bases de temps ne sont pas connues et/ou variables.

Tout d'abord, pour le partitionnement, l'approche la plus sûre est d'être robuste vis-à-vis du pire cas. Pour cela, il est toujours possible d'utiliser l'algorithme 2.1 comme en 2.5.2. Le calcul des taux d'utilisation doit être fait dans une base de temps unique, généralement le temps standard, pour le pire cas possible (les périodes les plus petites). Cela nécessite bien sûr d'être capable de caractériser les périodes des runnables dans le pire cas dans une seule base de temps. Par exemple, si certains runnables sont cadencés suivant le temps standard et d'autres runnables selon le régime moteur, on utilisera le régime moteur maximum pour avoir le pire cas.

Si les tâches séquenceurs sont cadencées sur des bases de temps indépendantes, quelle que soit la manière dont les tables de séquencements sont construites, n'importe quel slot d'une table est susceptible d'interférer avec n'importe quel slot d'une autre table pendant l'exécution. Cela signifie que, contrairement au cas synchrone étudié dans le reste de cette section, nous ne pouvons pas nous appuyer sur le séquencement des tâches plus prioritaires lors de la construction des tables d'ordonnancement des tâches séquenceurs suivantes. Ainsi, la manière la plus fiable et robuste de parer à tous les déphasages possibles entre les tâches revient donc à équilibrer au mieux la charge pour chacune des tâches, indépendamment de ce qui est fait pour les autres tâches. On applique donc les algorithmes de lissage de charge pour chacune des tâches séquenceurs sans tenir compte du reste, comme fait dans la section 2.4.

De plus, à cause des bases de temps variables à l'exécution, il n'est pas facile

de vérifier la faisabilité de la solution obtenue en utilisant les algorithmes, contrairement aux cas précédents où les slots surchargés se voient facilement. Cependant, s'il est possible de borner les vitesses d'ordonnancement des tâches séquenceurs, il devient alors possible d'utiliser le modèle multi-frame [44] pour vérifier la faisabilité. Comme expliqué précédemment, la transformation d'une tâche séquenceur en tâche multi-frame est directe et sans perte : les différents slots de la table d'ordonnancement deviennent les différentes instances de la tâche avec des temps d'exécution respectivement égaux à ceux des slots, et une valeur pessimiste de la deadline de chaque instance est donnée par la longueur du slot T_{tic} dans sa base de temps. Ainsi, en utilisant les vitesses maximums des tâches, nous pouvons utiliser les tests de faisabilité développés pour ce modèle [12, 70, 71].

2.6 Conclusion

Avec l'émergence des calculateurs multi-cœurs dans l'architecture électronique embarquée dans les véhicules, les méthodologies utilisées au moment de la conception ont donc besoin d'être adaptées afin de prendre en compte des nouveaux défis techniques. La conception de l'architecture logicielle de ces calculateurs fait partie de ces défis. Dans ce chapitre, nous avons présenté des solutions pratiques permettant d'ordonnancer un grand nombre de runnables adaptées au cas d'emploi le plus imminent pour les calculateurs multi-cœurs qui consiste à réduire la complexité de l'architecture électronique en utilisant des calculateurs multicœurs. Le problème de partitionnement des runnables sur les différents cœurs est résolu en appliquant une heuristique de type "worst fit decreasing" après avoir pris en compte leurs contraintes de localisation et de colocalisation. Nous adaptons ensuite l'algorithme "least loaded" utilisé pour les réseaux CAN afin de lisser la charge périodique induite par les runnables. Pour celui-ci, nous fournissons des garanties de performance dans le cas harmonique en dérivant une condition suffisante de faisabilité puis en donnant une borne sur le nombre de cœurs nécessaires pour exécuter un ensemble de runnables en respectant des contraintes d'échéances contraintes par la durée des slots des tâches séquenceurs. Des améliorations de cet algorithme sont présentés afin de traiter plus efficacement les ensembles de runnables avec des périodes non-harmonique et pour réduire les pics de charge. Enfin, nous décrivons comment appliquer ces algorithmes dans le cas où plusieurs tâches séquenceurs sont présentes sur un même cœur. Les algorithmes décrits dans ce chapitre se sont montrés efficaces dans les cas étudiés au niveau du lissage de la charge et de l'utilisation de la puissance disponible et s'adaptent facilement à d'autres usages comme la conception incrémentale par exemple. Les résultats présentés dans ce chapitre ont été publiés dans [47] et dans [50] une version étendue dans [5]. L'évaluation d'un outil permettant de calculer les valeurs de temps d'exécution des runnables nécessaires pour effectuer le lissage est présentée en Annexe B.

Dans ce chapitre, nous avons proposé une approche permettant d'optimiser l'utilisation de calculateurs multi-cœurs tout en vérifiant le respect des échéances des composants logiciels qui y sont exécutés. Cette approche respecte les choix de conception utilisées traditionnellement par les contructeurs automobiles. Dans le contexte des contraintes temporelles de bout-en-bout, le pendant de l'ordonnancement des tâches sur les calculateurs est l'ordonnancement des messages transmis sur les réseaux de communication, généralement de type CAN. Les transmissions des trames sur les bus CAN sont faits de manière asynchrones par les calculateurs. Pour cette raison, nous nous intéressons dans les chapitres suivant aux distributions des temps de réponse des trames CAN. Celles-ci peuvent en effet permettre de vérifier le respect de contraintes temporelles exprimées de manière probabiliste. Nous proposons une nouvelle approche de simulation dans le chapitre 3 et une nouvelle approche analytique 4 afin de les obtenir.

Chapitre 3

Simulation des systèmes de communication embarqués dans les véhicules

Résumé Dans cette partie, nous nous intéressons à l'étude des systèmes de communication embarqués dans les véhicules et plus particulièrement aux réseaux CAN, standard le plus répandu dans ce domaine. Durant les premières étapes de développement de l'architecture électronique d'un véhicule, deux approches sont principalement utilisées afin de vérifier les contraintes temporelles et dimensionner ces réseaux de communication : l'analyse pire cas et la simulation. Dans cette partie, nous illustrons le fait que ces deux approches sont complémentaires et que, dans la plupart des cas, il est utile de recourir aux deux approches de façon complémentaire. La contribution présentée ici est une approche par simulation prenant en compte les dérives d'horloges que subissent les nœuds de communication pendant leur fonctionnement. Nous évaluons à quel point les résultats obtenus par cette approche sont pertinents et utiles aux concepteurs pour la validation des systèmes de communication de type CAN. Pour cela nous nous intéressons dans un premier temps à des simulations de courtes durées autour des pires cas et dans un deuxième temps à des simulations de plus longues durées et aux distributions de temps de réponse obtenues. En pratique, nous verrons que les conditions initiales de simulation ainsi que la valeur des dérives d'horloge, n'ont pas d'impact significatif sur les distributions de temps réponses observées pour de grandes durées de simulation. En conséquence, effectuer une longue simulation avec dérive d'horloges est pertinent pour évaluer les performances temporelles d'un réseau CAN. Enfin, nous montrerons que les résultats obtenus de cette manière donnent des résultats similaires à l'approche traditionnellement utilisée et consistant à réaliser un grand nombre de courtes simulations avec des déphasages initiaux différents entre les calculateurs.

Sommaire

3.1 De l'intérêt de la simulation dans la conception des systèmes de communications automobiles

Lors de l'étude des contraintes temporelles sur un réseau de communication, l'analyse pire cas classique est peu représentative du comportement moyen des trames émises sur le réseau. Nous nous intéressons donc à une approche par simulation pour laquelle nous prenons en compte le phénomène de dérive d'horloge qui a été négligé dans les études sur les réseaux CAN menées jusqu'à présent. Nous verrons que prendre en compte ces dérives d'horloge est plus réaliste et permet aussi d'étudier en une seule simulation un large panel de déphasages différents entre les calculateurs connectés sur le même bus de communication.

3.1.1 Contexte

Les architectures électroniques embarquées dans les automobiles sont définies en amont pour un ensemble de véhicules pendant une période de trois à cinq ans et leurs propriétés temps-réel doivent donc être évaluées et validées très tôt dans le processus de conception. Parmi leurs nombreuses préoccupations, les ingénieurs responsables de la conception sont en particulier concernés par deux objectifs principaux :

- garantir la sûreté du système,
- dimensionner au plus juste l'architecture électronique pour trouver un bon compromis entre les coûts et les performances

Dans cette mesure, deux approches sont principalement utilisées lors des premières phases de conception des systèmes de communication embarqués dans les véhicules : l'analyse pire cas et la simulation. CAN étant le principal standard de communication entre calculateurs utilisé dans les voitures, les temps de réponses des trames CAN ont donc été l'objet de nombreux travaux de recherche au sujet de leur analyse ou leur simulation. Ces études se sont tout d'abord intéressées au pire temps de réponse (Worst Case Response Time) afin de répondre aux besoins de sûreté. Ensuite, ces travaux ont commencé à explorer les distributions de temps de réponses car cela donne une vision plus complète du comportement temps-réel du système. Comme le scénario pire cas pour le temps de réponse d'une trame est extrêmement rare [51], d'autres métriques liées aux distribution de temps de réponse, telles que les quantiles élevés (99,99 % par exemple), semblent plus pertinents pour arbitrer les décisions liées au dimensionnement de l'architecture électronique : les valeurs des temps de réponses correspondants aux quantiles sont en effet significativement plus basses et garantissent un niveau de sûreté satisfaisant pour la plupart des trames. Ainsi, l'obtention de distribution de temps de réponses précises ouvrirait la voie à des progrès significatifs pour la réduction des coûts matériels.

3.1.2 Travaux existants

L'analyse temporelle des réseaux CAN a déjà fait l'objet de nombreux travaux scientifiques. Des bornes sur les pires temps de réponses des trames ont d'abord été données dans [67] et [18]. Par la suite, certaines limitations matérielles ou liées à la pile de communication auparavant ignorées ont commencé à être étudiées [35, 17, 33]. En parallèle, des travaux sur les effets du trafic apériodique sur les temps de réponses [34] ou les conséquences de perturbations transitoires [54] ont été réalisés. De plus, des méthodes pour réduire les temps de réponses ou les rendre plus prédictibles ont également été explorées, comme par exemple dans [24] et [48].

Dans les travaux existants jusqu'à présent, certaines études reposaient sur de l'analyse probabiliste ou donnaient des distributions de temps en réponses résultant soit de l'ensemble des situations de déphasages entre calculateurs possibles [68, 69] soit du mécanisme de bit-stuffing du protocole CAN [57]. Cependant, ces travaux ne tenaient pas compte du phénomène de dérive d'horloge, alors que ce sujet est par exemple étudié dans le domaine des réseaux de capteur sans-fil [19].

En parallèle de ces travaux essentiellement analytiques, des outils des simulation ont été développés. Il existe plusieurs outils propriétaires permettant de simuler le fonctionnement d'un réseau CAN dont notament Symta/S [66], ChronSim [28] et enfin RTaW-Sim [63]. Au début des travaux de thèse, seul ChronSim supportait les dérives d'horloge via un module nommé ChronBus. Pour notre approche, nous avons utilisé et étendu l'outil RTaW-Sim comme détaillé en 3.2.3.

3.1.3 Pourquoi simuler les dérives d'horloge ?

Jusqu'à présent les pratiques de simulation pour les réseaux CAN consistaient à simuler une messagerie constituée de trames périodiques et en faisant l'hypothèse que les horloges des calculateurs émettant les messages étaient identiques, ce qui revient à simuler un système périodique. Les périodes des trames émises sur un bus CAN ont typiquement les propriétés de varier entre 5 ms et 2 s ainsi que d'être un multiple de 5 ms et diviser 2 s. En conséquence, la période du système (le PPCM des périodes des messages), appelée hyperpériode, est généralement de l'ordre de 2 s. Alors, de même manière que pour l'ordonnancement de tâche périodiques, l'ordonnancement des trames finis par se répéter après un certain temps donné et borné par $O_{max} + 2 \cdot T_{bus}$ [23] avec O_{max} la date la plus grande d'activation parmi les calculateurs communicants et T_{bus} l'hyperpériode. Ainsi, en considérant une situation initiale définie par les dates d'activation des stations émettrices, les temps de réponses observés finissent par se répéter après quelques secondes et sont directement et seulement dépendant des déphasages initaux entre les calculateurs. En procédant ainsi, réaliser une simulation ne permet d'observer qu'un petit nombre de temps de réponses différents pour chaque trame.

Les dérives d'horloges sont un résultat des différences de fréquences des horloges

des calculateurs responsables de l'instantiation des trames CAN. Du fait des défauts inhérents à leur processus de fabrication, les oscillateurs cadençant les horloges des calculateurs ne sont pas parfaitement identiques. De plus, leur fréquence peut également varier à cause de paramètres de l'environnement dans lequel ils fonctionnent comme en particulier la température. En pratique, ce phénomène est inévitable et a pour conséquences de faire varier sans cesse les déphasages entre les calculateurs communicants sur un réseau CAN. En conséquence, les temps de réponses des trames varient également au cours du fonctionnement du système. Lancer une simulation avec des dérives d'horloges permet ainsi d'observer un nombre important de temps de réponses distincts pour une même trame.

Afin d'illustrer cette importante différence, la figure 3.1 montre la répartition des temps de réponses obtenus pour une trame émise sur un bus CAN après une simulation sans dérive d'horloge et à différents temps de la simulation avec des valeurs fixes de dérives d'horloge à partir d'une même configuration initiale. Les barres verticales représentent la proportion de trames dont les temps de réponse sont compris dans différents intervalles de temps. Les intervalles de temps considérés sont de 0,5 ms et leurs valeurs centrales sont indiqués sur l'axe des abscisses. Nous observons que sans dérive (les barres en vert clair), les temps de réponses sont regroupés autour de deux valeurs respectivement proches de 13 s et 30 s. Plus exactement, même si le graphique ne le montre pas, les temps de réponses de cette trame pour cette simulation sans dérive d'horloge sont répartis de manière égale entre deux valeurs exactes. En revanche, lorsque les dérives sont prises en comptes, nous pouvons observer qu'au fur et à mesure que la simulation progresse, la distribution des temps de réponse évolue dans le temps. Ainsi, les temps de réponse, qui sont fortement regroupés autour de quelques valeurs au début de la simulation, finissent par se répartir au fur et à mesure que la simulation progresse.

Cette expérience montre clairement les différences qualitatives entre les simulations avec et sans dérive d'horloge. Ainsi, il est intéressant de remarquer qu'une seule simulation avec des dérives d'horloges permet d'obtenir une distribution de temps de réponses comprenant un important nombre de valeurs à partir d'un seule situation de départ. En effet, pour obtenir des distributions de temps de réponse, l'approche la plus répandue consiste à combiner les statistiques de temps de réponse résultant d'un nombre important de simulations réalisées à partir de paramètres initiaux (les déphasages entre calculateurs) différents. Au regard de ces importantes différences, et comme le phénomène de dérive est inévitable en pratique, il est important d'étudier plus précisément quels types de résultats nous obtenons en modélisant le phénomène de dérives d'horloge et comment nous pouvons utiliser cette approche pour aider à la conception des messageries embarquées dans les véhicules.

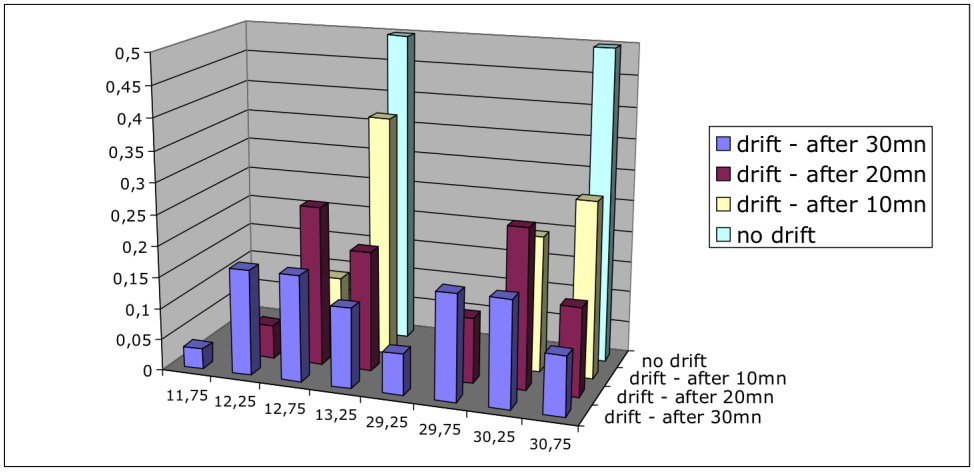

Figure 3.1 – Comparaison des distributions des temps de réponse observés pour une trame pour des simulations avec et sans dérive d'horloge [51]. Les différents couleurs correspondent aux différentes distributions. Les valeurs en abscisses sont les valeurs au centre des intervalles de temps de réponse.

3.1.4 Contributions

Dans ce chapitre, nous proposons donc une approche par simulation dans l'objectif d'étudier les temps de réponses sur les réseaux de type CAN. La particularité de cette approche consiste à modéliser le phénomène de dérive d'horloge ce qui a pour conséquence de permettre d'observer beaucoup plus de temps de réponse différents pour chaque trame à partir d'une seule simulation. En effet, sans dérive d'horloge, et comme nous étudions des trames périodiques, l'hyperpériode du système est assez courte. En conséquence, une trame avec une période égale à 50 ms sera émis 40 fois pour une hyperpériode de 2 s, et observera probablement un nombre encore inférieur de temps de réponses différents.

L'objectif des travaux présentés ici est de proposer une approche par simulation des réseaux CAN avec dérive d'horloge et d'étudier comment elle peut être utilisée et quels types de résultats elle permet d'obtenir. En se plaçant dans le contexte de la conception d'un réseau CAN pour un véhicule, nous cherchons donc à répondre aux questions suivantes :

- Comment modéliser les dérives d'horloge pour les intégrer à un simulateur de réseau CAN ?
- Que se passe-t-il autour des temps de réponse pire cas dans le cas de dérive d'horloge ?
- Quelle influence ont les paramètres de simulations sur les distributions de temps de réponse obtenues pour de grandes durées de temps simulé ?

Le chapitre est donc organisé de la manière suivante. En 3.2, nous détaillons le modèle et les hypothèses utilisées pour simuler le réseau CAN puis l'approche empruntée ainsi que l'outillage logiciel. L'objectif des travaux est de mettre en oeuvre une approche par simulation avec dérive d'horloge puis de l'utiliser pour répondre aux besoins des concepteurs automobile. Ainsi, nous nous intéressons tout d'abord aux "scenario pire cas" en 3.3. Pour ceux-ci, nous regardons dans quelle mesure il est possible de les trouver par simulatio. De plus, nous utilisons la simulation pour observer ce qu'il se passe juste après qu'on ait reproduit un pire cas avec les dérives d'horloge. Dans la partie suivante 3.4, nous cherchons à évaluer la valeur des distributions de temps de réponses obtenus pour des simulations de longue durée. En particulier, nous étudions l'influence des différents paramètres de simulation sur les statistiques finales et nous comparons nos résultats avec ceux obtenus sans dérive d'horloge.

3.2 Simulation des réseaux CAN avec dérive d'horloge

Dans cette section sont présentés le modèle utilisé pour la simulation ainsi que l'outil de simulation. Plus précisément, le modèle d'horloge et les hypothèses faites sur les noeuds CAN sont décrits, ainsi que le type de résultats obtenus par cette méthode.

3.2.1 Modélisation du réseau CAN

Dérives d'horloge

Dans l'architecture électronique des voitures, les calculateurs sont généralement cadencés par des oscillateurs à quartz. S'il est parfois envisagé d'utiliser des oscillateurs céramiques pour des applications peu critiques, ils ne remplissent généralement pas les exigences de tolérance sur l'imprécision maximale de la dérive pour être utilisés dans des bus communication. En ce qui concerne les oscillateurs à quartz, les variations de fréquences sont la conséquence de plusieurs facteurs dont les principaux sont :

- les défauts de fabrication
- le vieillisement
- la température

L'humidité et les vibrations peuvent également influencer la dérive d'un quartz mais de façon négligeable par rapport à la température et la qualité de fabrication.

La dérive est mesurée en *ppm* pour "parties par million" et exprime la différence de vitesse comparée à une horloge idéale. Ainsi, 1 *ppm* correspond à une déviation

de 1 μs toutes les secondes. Les valeurs typiquement données par les fournisseurs sont les tolérances suivantes :

- +/- 50 ppm dues aux défauts de fabrication
- +/- 5 ppm par année d'utilisation dues pour le vieillissement
- +/- 150 ppm dues à la température sur la plage de température de fonctionnement (*i.e.*, -40°C/125°C).

Dans notre cas, nous avons choisi un modèle de dérive simple basé sur des déviations fixes de vitesse d'horloge (positives ou négatives) par rapport à la vitesse nominale de fonctionnement du bus. En effet, nous faisons l'hypothèse que les variations dues au vieillissement des quartz sont négligeables pour les durées de simulation considérées. De plus, nous faisons l'hypothèse que les noeuds de communication fonctionnent à température constante. Pour une horloge c donnée cadençant un calculateur connecté à un réseau CAN, son temps local t_c, par rapport au temps global t, est donc donné sous la forme d'une fonction affine comme suit :

$$t_c(t) = \phi_c + \delta_c \cdot t$$

avec ϕ_c la date de démarrage du calculateur (son déphasage) dans le référentiel temporel du bus, et δ_c la valeur de la dérive. Par exemple, un taux de dérive de +50 ppm correspond à $\delta_c = 1.000050$. Pour ce travail, on assigne à chaque calculateur j connecté au réseau CAN une horloge définie par le couple (ϕ_j, δ_j). Sauf indication contraire, les valeurs de dérives utilisées par la suite sont choisies aléatoirement et uniformément dans une fourchette de tolérance précisée à chaque fois, donc entre -1,00150 et +1,000150 pour une tolérance de +/- 150 ppm.

Hypothèse sur le système de communication CAN

La figure 3.2 donne un modèle simplifié de l'architecture de la pile de communication CAN dans un calculateur automobile. Pour ce travail, nous ne considérons pas d'autres caractérisques matérielles particulières à part les dérives d'horloge. Dans chaque contrôleur de communication, les messages sont triés dans une file d'attente en fonction de leur priorité et il y a toujours un nombre suffisant de buffers de communication de manière à ce que le message le plus prioritaire soit toujours pris en compte pendant l'arbitrage. Enfin, nous négligeons les temps de copie depuis la file d'attente vers les buffers de communication.

Les trames CAN sont toutes supposées périodiques. Un message CAN m_i est défini par (T_i, O_i, C_i, P_i) avec T_i sa période, O_i son offset initial, C_i son temps de transmission et P_i sa priorité. Les offsets permettent de désynchroniser les émissions des trames envoyées par un même noeud afin de réduire les situations de surcharge temporaire et réduire les temps de réponse comme il est désormais pratique courante dans les réseaux de communication automobiles (voir par exemple [24]). Dans le

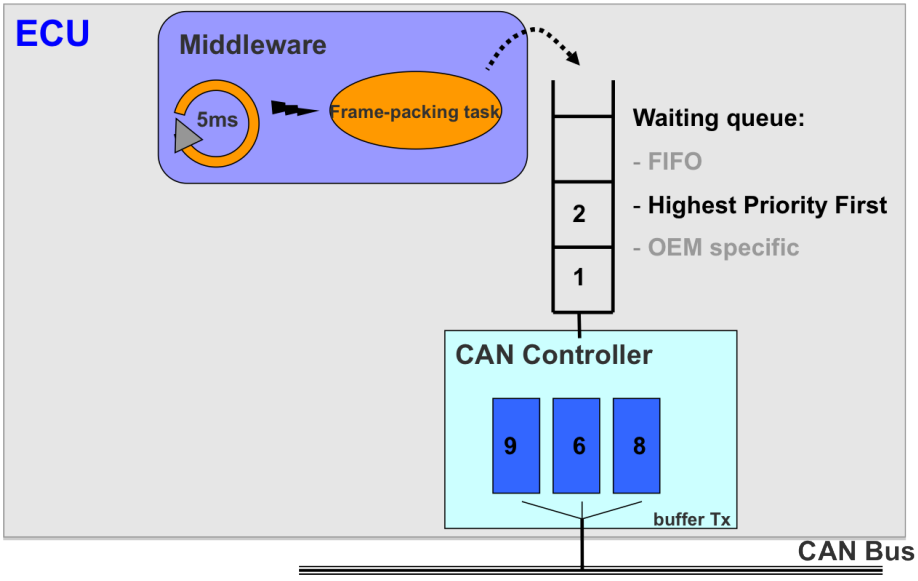

Figure 3.2 – Modèle simplifié de la pile de communication CAN [51]. Au niveau du logiciel middleware (AutoSar), une tâche périodique (de 5ms dans cet exemple) est responsable de la préparation des trames CAN à partir des données à transférer (les signaux). Une fois constituées, ces trames sont alors mises dans une file d'attente avant de pouvoir être copiées dans les buffers de transmission du contrôleur CAN qui est connecté au bus.

contexte de ces travaux, nous considérons systématiquement le pire cas de bit-stuffing pour C_i. Soit s_i, le nombre d'octets utiles, et τ_{bit} le temps de transmission d'un bit de donnée sur le bus :

$$C_i = (\left\lfloor \frac{34 + 8 \cdot s_i}{4} + 47 + 8 \cdot s_i \right\rfloor \tau_{bit}$$

Comme l'instantiation des trames CAN est cadencée par l'horloge de leur émetteur, les quantités T_i et O_i sont exprimées par rapport à l'horloge locale de leur émetteur. Ainsi, des messages ayant des périodes égales mais envoyés par différents émetteurs auront des périodes de transmission légèrement différentes dans le référentiel temporel du bus de communication. Par contre, C_i, le temps de transmission sur le bus CAN, ne dépend que de la quantité de données utiles transmises par la trames (de 1 à 8 octets) et correspond donc à une durée exprimée dans le référentiel temporel du bus CAN. Simuler le système revient finalement à calculer les dates d'arrivée de chacune des trames puis reproduire les processus d'arbitrage et de transmission du protocole.

3.2.2 Détermination des distributions de temps de réponse

Les temps de réponse des trames sont une conséquence des caractéristiques des trames et des déphasages entre les calculateurs. Grâce à des outils d'analyse tel NETCAR-Analyzer [1], il est possible d'obtenir des bornes supérieures sur les temps de réponses correspondant au pire cas, valables quels que soient les déphasages entre les calculateurs. La simulation est cependant plus adaptée pour évaluer les cas moyens. L'originalité de notre approche est de modéliser explicitement les dérives d'horloge dans le simulateur. Les approches de simulation classique consistent à collecter les statistiques d'un certain nombre de scénarios correspondants à plusieurs configurations initiales distinctes (*i.e.*, des déphasages initiaux différents entre les calculateurs). Cela est nécessaire car chaque simulation ne sera capable de capturer qu'un nombre limité de temps de réponse pour chaque trame, comme la période du système (le *ppcm* des périodes des trames) est relativement faible (typiquement quelques secondes). Notre approche, par contre, consiste à collecter des statistiques pour un seul scénario en simulant pendant une grande durée (donc une seule simulation) avec des dérives d'horloge. Comme les déphasages entre les calculateurs varient continûment à cause des dérives d'horloges, une seule simulation est suffisante pour observer un nombre très important de temps de réponses différents pour chaque trame. Comme nous ne réalisons qu'une seule simulation, cette approche est plus pratique à mettre en oeuvre au point de vue expérimental. De plus, cette approche est bien entendu plus réaliste dans la mesure où elle tient compte des dérives d'horloge qui sont inévitables dans la pratique. Notre objectif est de mettre en évidence dans quelle mesure les résultats obtenus avec notre approche sont pertinents et utilisables pour les concepteurs dans leur tâche d'évaluation des performances d'un réseau de communication de type CAN.

3.2.3 Outillage logiciel

L'outil logiciel que nous utilisons, RTaW-Sim [63], est un simulateur à événements discrets pour le bus CAN et produisant les distributions de temps de réponse des trames CAN. Ce logiciel est gratuit et disponible en téléchargement. La granularité de la simulation est de l'ordre de $1\mu s$, ce qui permet de modéliser précisément le fonctionnement d'un bus CAN au niveau des bits d'information même dans la vitesse maximale autorisée par le standard qui est de 1 Mbit/s. Ce logiciel est doté d'autres fonctionnalités telles que la modélisation des buffers de communication et des mécanismes d'injection de fautes que nous n'utiliserons pas dans le cadre de ce travail. Enfin, nous utiliserons aussi une fonctionnalité qui permet de rejouer les scénarios provoquant des pire cas pour des trames de notre choix [2] trouvés avec

1. NETCAR-Analyzer est un outil d'analyse pire cas pour les réseaux CAN [61] ©INRIA/INPL
2. Pour chaque trame, la configuration de déphasage initiaux menant à un pire cas est généralement différente.

Netcar-Analyzer [61]. L'algorithme de calcul du pire cas avec offsets de Netcar-Analyzer est propriétaire mais des résultats ont depuis été publiés[42].

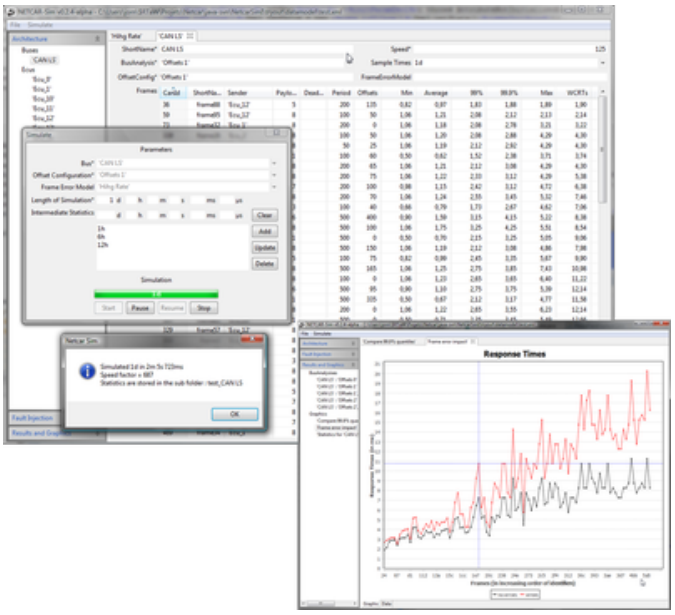

Figure 3.3 – *Capture d'écran de RTAW-Sim. La grande fenêtre en arrière plan contient une description des propriétés des trames. Les deux petites fenêtres monrent le choix des paramètres de simulation et l'avancement de la simulation. La dernière fenêtre en bas à droite montre des statistiques de temps de réponse obtenues après une simulation.*

Dans ce chapitre, nous présentons des résultats expérimentaux obtenus pour des ensembles de messages issus de benchmarks ou issus de messageries existantes. Le simulateur est capable de simuler environ 700 kilo-événements par seconde sur un processeur à 2GhZ, ce qui signifie qu'il est possible de simuler 24 heures de communication sur un bus CAN avec une charge de 60 % en environ 20 minutes. En comparaison, l'approche généralement utilisée (voir §3.2.2) consistant à simuler le plus de scénario différents possibles pour quelques hyperpériodes se traduit souvent par des heures de calcul et ceci même avec des modèles plus grossiers.

3.3 Etude du pire cas

Dans le but de garantir la sûreté, les concepteurs doivent se rapporter au pire temps de réponse des trames qui est généralement obtenu par analyse. Cependant, les techniques d'analyse ne nous renseignent pas sur la durée des scenarios où le

pire temps de réponse se produit ni à quelle fréquence ces situations se produisent.

3.3.1 Trouver le scénario pire cas

Trouver le pire temps de réponse par analyse est un problème pouvant être résolu par certains logiciels comme par exemple NETCAR-Analyzer [61]. En revanche, il est très difficile de reproduire le scénario pire cas par simulation sur un réseau de taille réaliste. Trouver le scénario pire cas pour une trame nécessite d'identifier la configuration de déphasage des calculateurs provoquant le pire temps de réponse possible.

Le nombre de configurations de déphasage entre calculateurs à étudier dans un réseau CAN est $(\frac{T}{\tau})^N$, avec T l'hyperpériode de l'ensemble des trames (le *ppcm* de leurs périodes), τ le temps-bit du bus (temps pour transmettre un bit sur le réseau) et N le nombre de calculateurs communicants sur le réseau. Pour un réseau CAN high-speed typique à 500 kbit/s constitué de 10 calculateurs émettant un ensemble de trames périodiques ayant une hyperpériode de 2 s, nous obtenons 10^{60} configurations de déphasages alternatives. Parmi elles, il est possible que plusieurs configurations de déphasage soient équivalentes mais le nombre de configurations donnant des temps de réponses différents reste très élevé. Du fait des dérives d'horloge, de nombreuses configurations de déphasage vont être explorées pendant une seule simulation. Cependant, cela ne garantit pas de trouver celle provoquant le pire temps de réponse d'une trame. Heureusement, il est possible d'utiliser les résultats des outils d'analyse afin d'identifier une configuration de déphasage entre calculateurs provoquant un pire cas de temps de réponse pour une trame donnée et de rejouer ce scénario. C'est ce que nous allons faire à présent.

3.3.2 Reproduire un scénario pire cas

Dans ces premières expériences, nous simulons les configurations de déphasage menant au pire temps de réponse d'une trame choisie et examinons ce qui se passe peu de temps après son occurrence. Dans cet objectif, nous démarrons la simulation en utilisant les paramètres de déphasage (les valeurs ϕ_i) entre calculateurs correspondant à un scénario pire cas et que nous avons obtenus par analyse avec Netcar-Analyzer. Pour cette expérience, nous avons utilisé la messagerie venant du benchmark SAE [4] qui est un réseau CAN à 250 kbit/s correspondant à une charge moyenne de 62,3%. Le message étudié ici est la trame avec l'identifiant 49, un des plus bas identifiants de l'ensemble de la messagerie. Les offsets entre trames émises par un même calculeur ont été calculés en utilisant Netcar-Analyzer. Les statistiques résultant de la simulation sont présentées dans les figures 3.4, 3.5 et 3.6. Nous avons réalisé quatre simulations correspondant à des paramètres différents de dérive d'horloge.

Dans un premier temps dans ce chapitre, nous nous intéressons à une vision globale de la messagerie et de l'évolution des statistiques pour l'ensemble des trames. Ainsi, l'ensemble des trames est donné sur l'axe des abscisses par ordre décroissant de priorité. La référence est représentée par les courbes noire et verte correspondant respectivement aux moyennes et maximums des temps de réponse des trames pour une configuration sans dérive d'horloge. En l'absence de dérive d'horloge et comme les durées de simulation sont plus longues que l'hyperpériode du système (quelques secondes sans dérives), les courbes n'évolueront pas au cours de l'expérience. Les autres courbes donnent les statistiques obtenues pour des valeurs aléatoires d'horloge dans des limites respectives de +/- 50 ppm et +/- 150 ppm (valeurs réalistes) et +/- 1000 ppm (valeur très pessimiste par rapport à la documentation des fournisseurs).

Le premier graphique montre qu'après 10 secondes de simulation (5 hyperpériodes), les temps de réponses moyens des trames de toutes les simulations (sauf +/- 1000 ppm) sont très proches des maximums obtenus sans dérive. Ceci indique que la plupart des temps de réponses sont assez élevés. Le second graphique montre qu'après 1 minute de temps simulé, les temps de réponse ont sensiblement baissé. Il convient de noter que plus les valeurs de dérive sont importantes, plus les moyennes des temps de réponses se sont éloignées des maximums (en diminuant) . Enfin, le dernier graphique montre qu'après 10 minutes de temps simulé, les courbes de moyennes de temps de réponses se stabilisent autour de valeurs qui sont significativement plus faibles que celles observées pour des temps simulés plus courts, quelles que soient les valeurs des dérives d'horloge.

Cette série d'expériences montre deux phénomènes. Tout d'abord, dans le cas de dérives d'horloge, les moyennes de temps de réponse des trames se stabilisent autour de valeurs communes comme l'indique la dernière figure 3.6. Cela se fait quelle que soit la valeur des dérives, même si cela est plus rapide si les dérives sont plus fortes. La courbe rouge correspondant à des valeurs de dérive plus faible (+/- 50ppm) nécessite encore un peu de temps pour rejoindre les autres. De plus, ces moyennes ont sensiblement baissé, ce qui indique que les temps de réponses des trames ne sont plus aussi grands qu'au début de l'expérience. Dans la mesure où 150 ppm est une valeur réaliste pour borner les dérives d'horloge, ces expériences indiquent que les trames commencent à avoir des temps de réponses proches du pire cas après 1 minute. Intuitivement, cela signifie que les dérives d'horloges qui semblent néfastes parce qu'elles réduisent la prédictabilité du système, peuvent être également bénéfiques car elles permettent d'écourter les situations où les déphasages entre calculateurs provoquent les pire temps de réponse.

3.3.3 Temps de réponse observé versus scénario pire cas

Dans cette deuxième série de simulations, nous étudions différents réseaux CAN sur une durée de 4 heures (*i.e.*, un long trajet en voiture) en utilisant 20 configurations initiales de déphasages choisies aléatoirement et des valeurs de dérives d'horloge comprises dans la fourchette +/- 150 ppm. Nous relevons alors le maximum des temps de réponse observés pendant ces simulations pour une trame de faible priorité ainsi que les temps de réponses moyens et deux grands quantiles. La valeur "pire cas" est obtenue en reproduisant le scénario pire cas comme fait précédement. Nous choisissons ici de nous intéresser aux trames de faible priorité car, d'une part, il est plus difficile de trouver leur pire cas par simulation et d'autre part, leurs temps de réponses sont plus sensibles aux variations des paramètres de

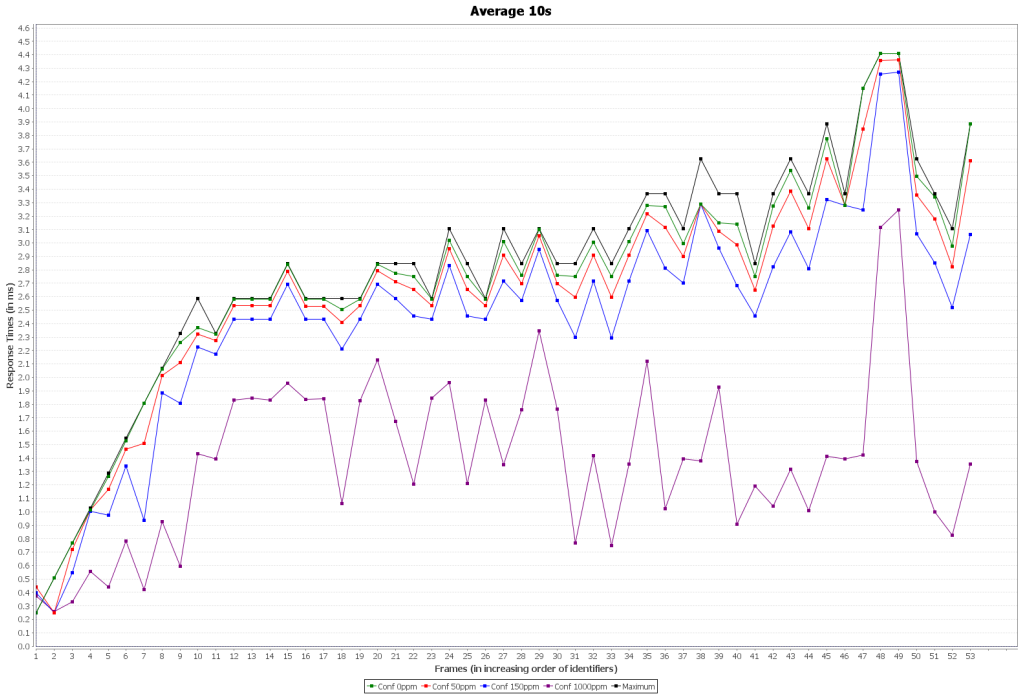

Figure 3.4 – *Statistiques obtenues par simulation à partir des déphasages menant au scénario pire cas pour la trame 49. Le graphique montre les temps de réponse après 10 s de temps simulé. Les trames triées par ordre décroissant de priorité se trouvent sur l'axe des abscisses. L'axe des ordonnées donnent les valeurs de temps des points des différentes courbes suivantes. La courbe noire donne les maximums observés. Les autres courbes correspondent aux moyennes des temps de réponse pour des configurations sans dérive (vert), ou des intervalles de tolérance de dérives de +/- 50 ppm (rouge), +/- 150 ppm (bleu) et +/-1000 ppm (violet).*

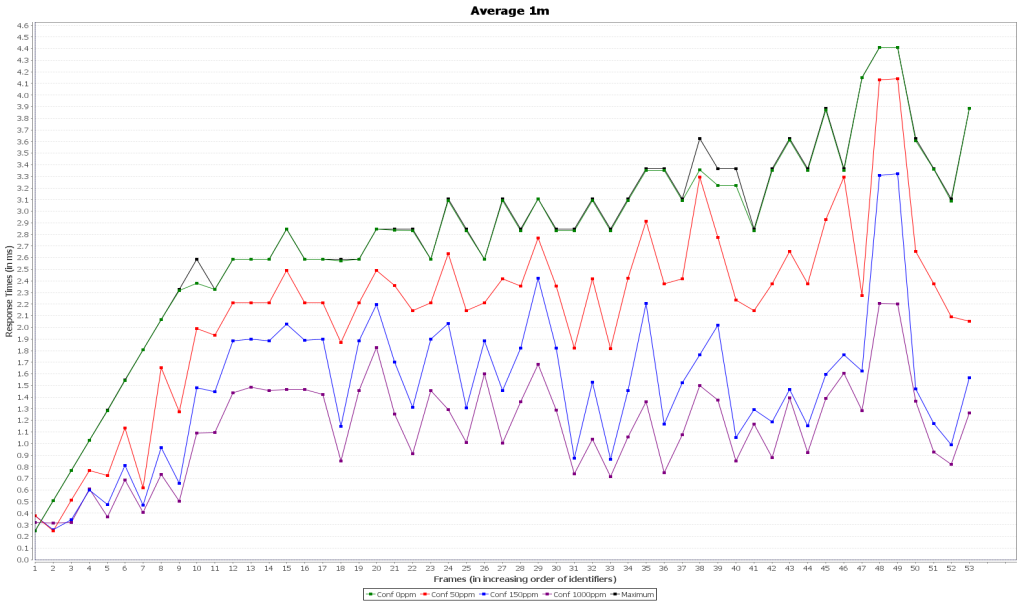

Figure 3.5 – *Voir légende de la figure 3.4. Le graphique montre les statistiques de temps de réponse après 1 mn de temps simulé.*

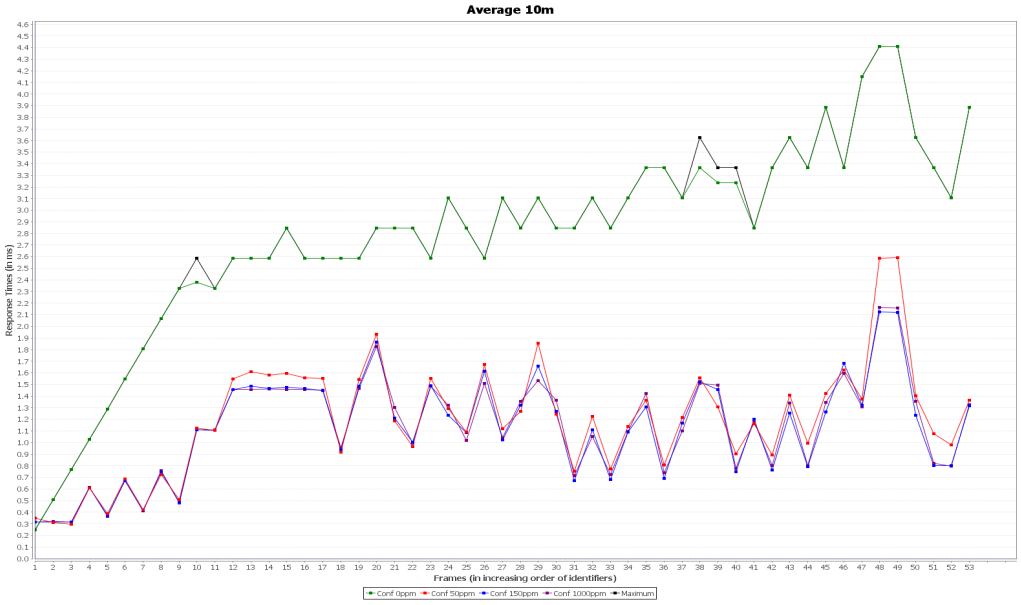

Figure 3.6 – *Voir légende de la figure 3.4. Le graphique montre les statistiques de temps de réponse après 10 mn de temps simulé.*

Réseau CAN	SAE	Zeng, et al.[68]	Cas Didactique
Nombre d'ECUs	6	6	18
Nombre de trame	53	69	101
Vitesse du bus (kbit/s)	250	500	125
Charge réseau	62,3%	60,25%	33,8%
Pire cas calculé analytiquement (ms)	4,408	4,414	**30,414**
Maximum (ms)	4,408	4,163	**10,195**
Quantile 99,9% (ms)	4,125	3,675	8,65
Quantile 99% (ms)	3,875	2,975	5,85
Moyenne (ms)	1,332	0,721	0,96

Table 3.1 – *Statistiques pour la trame avec le plus grand temps de réponse pour chaque benchmark, en utilisant des déphasages initiaux aléatoires entre les calculateurs et des valeurs de dérives dans la fourchette +/- 150 ppm. Les valeurs des quatre dernières rangées sont les plus grandes valeurs observées sur l'ensemble des expériences menées sur le réseau correspondant.*

simulation. En effet, les trames plus prioritaires seront beaucoup moins impactées car les modifications toucheront beaucoup moins de trames pouvant les bloquer (celles qui sont plus prioritaires).

Pour cette série de simulations, nous avons utilisé le benchmark SAE, la messagerie utilisée dans [68] et une version modifiée (pour des raisons de confidentialité) d'une messagerie utilisée chez PSA Peugeot-Citroën que nous appellons ici "cas didactique". Les deux premières messageries utilisées sont détaillées en annexe. Les résultats sont présentés dans le tableau 3.1.

On peut noter que le pire cas calculé avec Netcar-Analyzer a pu être obtenu par simulation pour le premier réseau mais il faut remarquer que le nombre de calculateurs communiquant sur ce réseau est faible, ce qui réduit considérablement le nombre de configurations de déphasages à explorer. Pour le second réseau, le plus grand temps de réponse simulé est assez proche du pire cas pour la même raison. Par contre, le troisième réseau a un nombre de calculateurs plus réaliste par rapport aux réseaux de communications embarqués dans les voitures produites actuellement. Les résultats obtenus pour ce dernier sont donc sensiblement différents. En effet, les valeurs moyenne et maximum, ainsi que les quantiles sont sensiblement plus faibles que le temps de réponse pire cas.

Cela confirme donc l'intérêt de travailler sur les statistiques de temps de réponse pour les réseaux automobiles et de regarder au-delà du scenario pire cas. En effet, celui ci est rare et provoque un temps de réponse très largement supérieur, même lorsque que nous considérons 99,9% des temps de réponse. Pour les réseaux simples en revanche, utiliser les dérives d'horloge permet souvent de tomber sur

le scenario pire cas au cours d'une simulation. Cette dernière remarque est intuitive dans la mesure où le nombre de configurations de déphasages à explorer est considérablement plus réduit.

3.4 Distributions de temps de réponses des messages

Dans cette partie, nous étudions dans quelle mesure les résultats obtenus par l'approche utilisée ici sont représentatifs de ce qu'il se passe sur le réseau. Plus particulièrement, nous étudions l'impact des paramètres de simulation sur les distributions de temps de réponses des trames afin de déterminer s'il est ou non pertinent de ne simuler qu'une seule trajectoire avec des dérives d'horloge pendant une longue durée. La raison pour laquelle nous utilisons de longues durée de simulation est d'observer les statistiques globales représentant le comportement du réseau sur de longs scénarios : un long trajet ou la durée totale de fonctionnement d'un véhicule (estimé à 7000 heures). Les statistiques obtenues par simulation indiquent alors quelles performances peuvent être attendues. Ceci doit permettre aux concepteurs de conclure sur l'adéquation entre la messagerie et le bus et donc valider leurs dimensionnements : choix du débit du bus et des caractéristiques des trames (priorité, offsets et périodes). Nous comparons également les résultats obtenus avec notre approche avec ceux obtenus par les approches existantes consistant à réaliser un grand nombre de simulations sans dérive d'horloge avec des scénarios de déphasages différents.

3.4.1 Impact de la durée de simulation

Ici nous étudions la stabilisation des statistiques dans le temps. En effet, nous avons observé lors de premières simulations que les statistiques de temps de réponses évoluent au début des simulations mais finissent par se stabiliser après un certain temps. Ceci est par exemple observable dans les figures 3.4, 3.5 et 3.6 pour la configuration +/- 1000 ppm dont les courbes sont différentes à 10s et 1 minute mais identiques à 1 minute et 10 minutes de simulation. En réalité, dans le cas de dérives constantes utilisées pour nos simulations, le système étudié est toujours périodique, même si l'hyperpériode est sensiblement plus importante que sans la dérive.

Supposons que l'hyperpériode des trames émises par un noeud i est T_i et que la valeur de la dérive subie par le noeud i est δ_i, alors l'hyperpériode observée sur le bus est alors $T_i \cdot \delta_i$, en accord avec le modèle de dérive fixe utilisé pour chaque calculateur. En procédant ainsi pour tous les calculateurs, l'hyperpériode du système, le ppcm de tous les $T_i \cdot \delta_i$, devient alors potentiellement très importante. Par exemple, en prenant des hyperpériodes proches d'une valeur T pour un réseau comprenant N noeuds et des valeurs de dérives proches d'une valeur δ, pour que

les produits $T \cdot \delta$ soient distincts,la période du système pourra être aussi importante que $(T \cdot \delta)^N$, si tous les $T_i \cdot \delta_i$ sont premiers entre eux. Pour des valeurs réalistes $T = 2s$, N=10 et des dérives proches de 150 ppm, nous obtenons alors une nouvelle hyperpériode d'environ $(2s \cdot 1.00015)^{10} \approx 17mn$. Après une période transitoire au démarrage du réseau due au démarrage décalé des calculateurs, l'ordonnancement des trames CAN devient périodique (pour une charge inférieure à 100%). En conséquence, la distribution des temps de réponse correspondant à la partie périodique où l'ordonnancement est répété indéfiniment deviendra prépondérante et les statistiques correspondant à la phase initiale deviennent de plus en plus négligeables au fur et à mesure que la durée de simulation augmente.

Cela signifie donc qu'il n'est pas nécessaire de simuler un réseau CAN au delà d'une certaine durée (qui dépend de chaque réseau) car les distributions de temps de réponses se stabilisent après un certain moment. En revanche, nous avons établi que les distributions de temps de réponses finissent par se stabiliser mais non qu'elles sont identiques. Ainsi, lors des prochaines expériences, nous comparons les distributions de temps de réponses obtenues pour des configurations de déphasages de départ différentes, et des valeurs de dérives différentes.

3.4.2 Impact des déphasages initiaux entre calculateurs

Dans cette série de simulations, nous étudions les distributions de temps de réponses obtenues pour de grandes durées de simulation à partir de configurations de déphasages différentes entre calculateurs. Nous présentons ici les résultats obtenus pour une durée simulée égale à 24 h. Nous utilisons pour cela la messagerie introduite dans [68] composée de 60 trames CAN émises par 6 calculateurs sur un bus avec un débit de 250 kbit/s correspondant à une charge totale de 60,25%. Les offsets initiaux des trames sur les calculateurs ont été calculés avec l'algorithme SOA de NETCAR-Analyzer [61].

Les figures 3.7, 3.8 et 3.9 donnent les statistiques des temps de réponses obtenues pour des valeurs égales de dérives d'horloges mais des déphasages initiaux différents. Ces valeurs ont été choisies aléatoirement. Les valeurs de dérives sont comprises dans l'intervalle +/- 150 ppm. Une première remarque est que les temps de réponses maximum observés varient légèrement d'une configuration à une autre, en particulier pour les trames de plus basse priorité (partie droites des graphiques). Cela est compréhensible car les maximums ne correspondent souvent qu'à une seule occurence de temps de réponse et sont très sensibles aux conditions initiales de la simulation. De manière prévisible, les temps de réponses minimum observés sont égaux d'une configuration à une autre car les déphasages entre les calculateurs varient de telle manière à ce que la situation où une trame est envoyée sans retard (car le bus est libre et aucune autre trame n'attend) arrive au moins une fois en 24 heures. En revanche, ce qui est plus surprenant est que les temps de réponses

moyens ainsi que les quantiles 99% et 99,9% sont presque identiques. En effet, il n'y pas de différence notable au niveau des moyennes. Seules les trames d'identifiant 45 et 55 montrent d'infimes différences pour les courbes du quantile 99%. Enfin, les différences pour le quantile 99,9% sont également assez subtiles au niveau des trames d'identifiante 27, 34 et 58. C'est un résultat significatif car cela indique que, pour de grands temps simulés, les déphasages initiaux antre calculateurs n'ont pas d'impact sensible sur les statistiques de temps de réponses obtenues (sauf bien sûr pour les maximums observés).

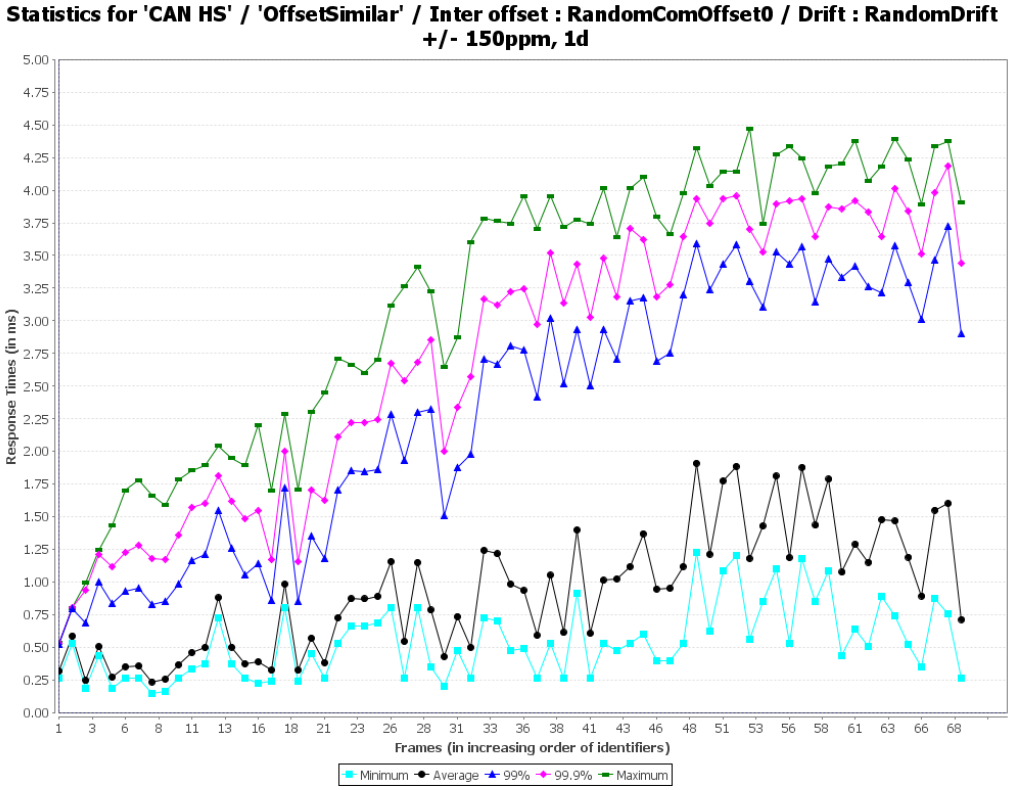

Figure 3.7 – *Statistiques de temps de réponse de la messagerie la première configuration de déphasages aléatoires. L'axe des abscisses représente les trames classées par ordre décroissant de priorité (i.e., par ordre croissant d'identifiant). L'axe des ordonnées représente les temps de réponses. Les différentes courbes représente respectivement du bas vers le haut : le minimum (cyan), la moyenne (noir), le quantile 99% (bleu), le quantile 99,9% (magenta) et le maximum (vert).*

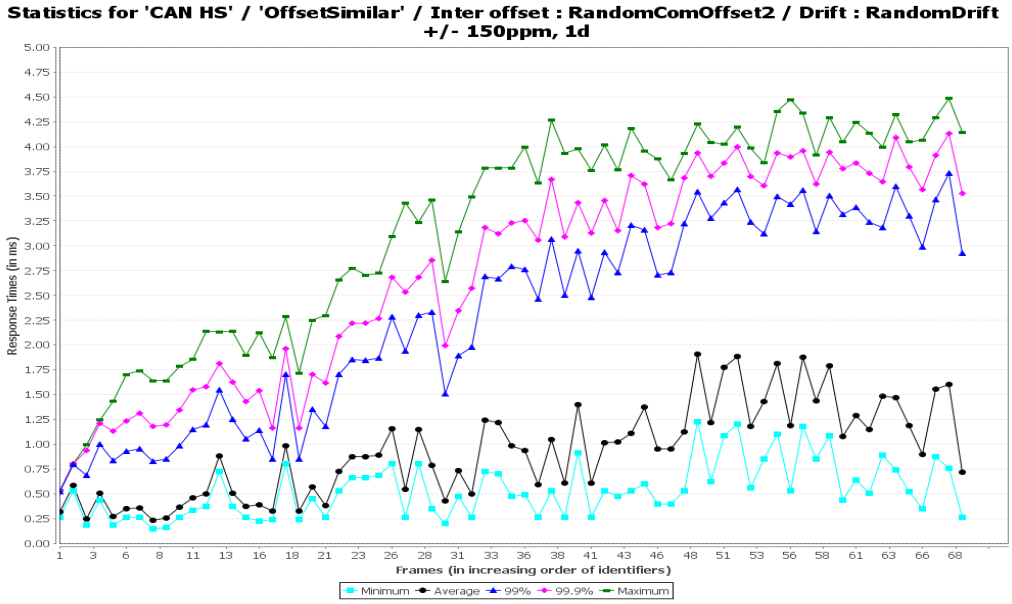

Figure 3.8 – *Statistiques de temps de réponse de la messagerie pour une seconde configuration de déphasages (même légende que la figure 3.7).*

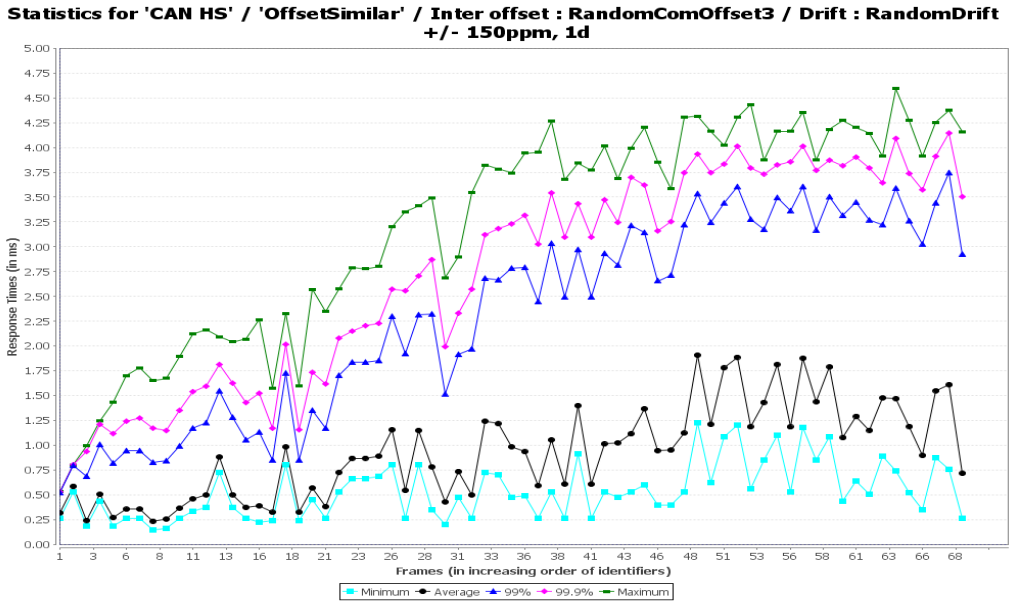

Figure 3.9 – *Statistiques de temps de réponse de la messagerie pour une troisième configuration de déphasages(même légende que la figure 3.7).*

3.4.3 Impact de la valeur des dérives

A présent, nous cherchons à évaluer l'impact des valeurs de dérives d'horloges sur les statistiques de temps de réponse des trames. Les premières expériences que nous avons menées à ce sujet semblaient indiquer que, de manière similaire aux configurations de déphasage initial, la valeur des dérives avaient un impact négligeable sur les statistiques observées à long terme. Cependant, utiliser des valeurs de dérives différentes induit des charges différentes sur le réseau, ce qui devrait intuitivement impacter les distributions de temps de réponse. En conséquence, nous avons alors cherché à utiliser des fourchettes plus larges pour les dérives d'horloge de manière à faire varier de manière plus significative la charge globale du réseau. Pour cela nous choisissons des valeurs proches les unes des autres tout en restant dans l'intervalle de tolérance. Par opposition aux expérimentations précédentes nous regardons à présent plus précisément la distribution de temps de réponse d'une trame en particulier.

La figure 3.10 présente les statistiques cumulées de temps de réponse de la trame la moins prioritaire du benchmarch SAE pour des configurations de dérives d'horloges différentes simulées pendant 24 heures. La durée de simulation est choisie de manière à s'assurer que les statistiques se sont stabilisées. Les courbes mauve et verte correspondant à des configurations où les valeurs des dérives ont été choisies aléatoirement respectivement dans les intervalles +/- 1 ppm (quasiment pas de dérive) et +/- 1000 ppm (dérive potentiellement importante). Les courbes bleue et rouge correspondent à des configurations où les valeurs des dérives d'horloge ont été choisies proches mais différentes les unes des autres (pour éviter que les stations soient synchrones) de manière à ce qu'elles soient aux limites des seuils de tolérance de l'intervalle +/- 1000 (c'est-à-dire *-1000 ppm*, *-999 ppm*, *-998 ppm*, etc. pour la courbe rouge et *+1000 ppm*, *+999 ppm*, *+998 ppm*, etc. pour la courbe bleue).

Sur la figure, nous observons que les statistiques de temps de réponses diffèrent d'une configuration à une autre confirmant ainsi que la valeur des dérives d'horloge a un impact sur les temps de réponse. En effet, choisir des dérives proches de +1000 ppm ou -1000 ppm induit une différence de 0,2% sur la charge réseau ce qui se traduit par une différence de temps de réponse, en particulier pour les trames de faible priorité. Comme la charge globale sur réseau est plus faible pour la courbe rouge, celle-ci est au dessus des autres jusqu'à 2,5ms car on observe plus de petits temps de réponse. La courbe bleue se comporte de manière opposée pour les mêmes raisons (moins de faibles temps de réponse et plus de grands temps de réponse). En revanche, les courbes verte et mauve sont proches l'une de l'autre ce qui s'explique par le fait que leurs charges réseau sont assez proches car leurs dérives d'horloges ont été choisies aléatoirement et de manière uniforme autour d'une même valeur moyenne.

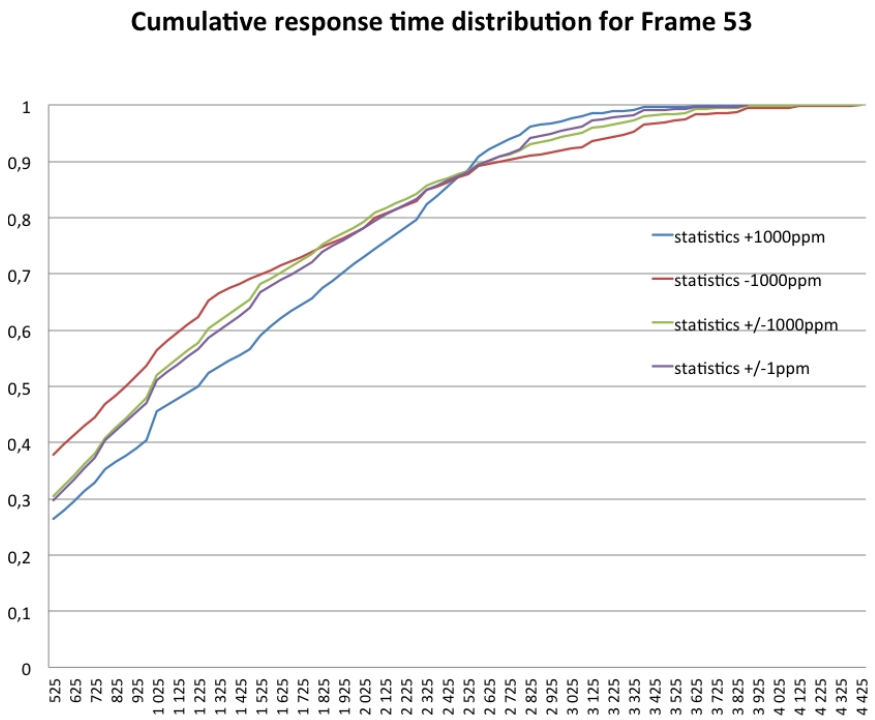

Figure 3.10 – *Statistiques cumulées de temps de temps de réponse pour la trame 53 du benchmark SAE pour différentes configurations de dérives d'horloges simulées pendant 24h. L'axe des abscisses donnent les temps de réponse en μs et l'axe des ordonnées représente les statistiques cumulées.*

Ces simulations montrent que les dérives d'horloge influencent les distributions de temps de réponse des trames. Pour mettre cela en évidence, nous avons dû recourir à des valeurs plus importantes que celles tolérées dans les réseaux embarquées dans les voitures. La prochaine série de simulation vise à étudier ce qu'il en est dans les cas réalistes rencontrés dans l'industrie pour lesquels les valeurs des dérives d'horloges sont typiquement plus faibles.

3.4.4 Cas avec des bornes de dérives réalistes

Cette dernière expérience vise à évaluer l'impact des dérives d'horloge avec des paramètres correspondant à des cas plus réalistes. Dans la mesure où les déphasages initiaux entre les calculateurs ne peuvent pas être prédits et en considérant des dérives d'horloge faibles, nous cherchons à évaluer la pertinence de l'approche consistant à ne faire qu'une seule longue simulation d'un bus CAN pour étudier les

performances de ce réseau.

Dans cette perspective, nous comparons les distances entre les distributions obtenues par simulation. Les déphasages initiaux entre calculateurs sont choisis aléatoirement. De manière similaire, les valeurs des dérives d'horloge sont choisies aléatoirement dans des intervalles de tolérances pré-établi : +/-100 ppm et +/-1000 ppm. Chaque série expérimentale est constituée de 10 simulations de 48h de temps simulé pour le benchmark SAE. Nous comparons alors les statistiques de temps de réponse de la trame de plus faible priorité à différents instants de la simulation. À cet effet nous procédons comme suit : pour la trame considérée, nous divisons l'intervalle des temps de réponses observés en 100 sous-intervalles. Ensuite, nous calculons l'écart type du nombre de trames ayant un temps de réponse compris dans les intervalles pour les dix simulations et pour chacun des intervalles avant d'en réaliser la moyenne. Ainsi, plus les résultats des simulations sont proches, plus la valeur calculée sera faible. La figure 3.11 donne les résultats obtenus par cette méthode.

Le graphique montre clairement que la moyenne des écart-types des temps de réponse diminue sensiblement au fur et à mesure que les simulations progressent. Une autre remarque intéressante est que ce phénomène est plus marqué pour la série de simulations avec une tolérance de +/- 1000 ppm que pour les dérives d'horloges pour lesquelles les écart-types sont plus faibles. En combinant ces observations avec le fait que les simulations se stabilisent dans le temps, nous pouvons conclure que les statistiques de temps de réponse convergent vers une distribution limite quels que soient leurs déphasages initiaux et les dérives. Ce phénomène est d'ailleurs encore plus marqué pour les trames de haute priorité pour lesquelles les moyennes des écart-types sont encore plus faibles.

Cela suggère que, pour des valeurs réalistes de dérive d'horloge, il est possible de décrire les performances d'une trame émise sur un réseau CAN avec une seule distribution limite. Cela indique également qu'il est suffisant de ne faire qu'une seule longue simulation pour évaluer les performances d'un réseau CAN sur une grande période de fonctionnement.

De plus, d'autres expériences montrent qu'utiliser des valeurs de dérives d'horloge artificiellement plus grandes que celles rencontrées en pratique (mais aléatoires), permet d'accélérer la stabilisation des statistiques des temps de réponses. Ceci peut permettre de réduire la durée des simulations lorsque l'on cherche à obtenir les performances d'un réseau CAN sur une grande fenêtre de fonctionnement.

En revanche, pour des temps simulés plus court (moins de 10 minutes), il convient de respecter aussi fidèlement que possible les conditions réelles car il n'est pas garanti que les distributions de temps de réponse se soient stabilisées.

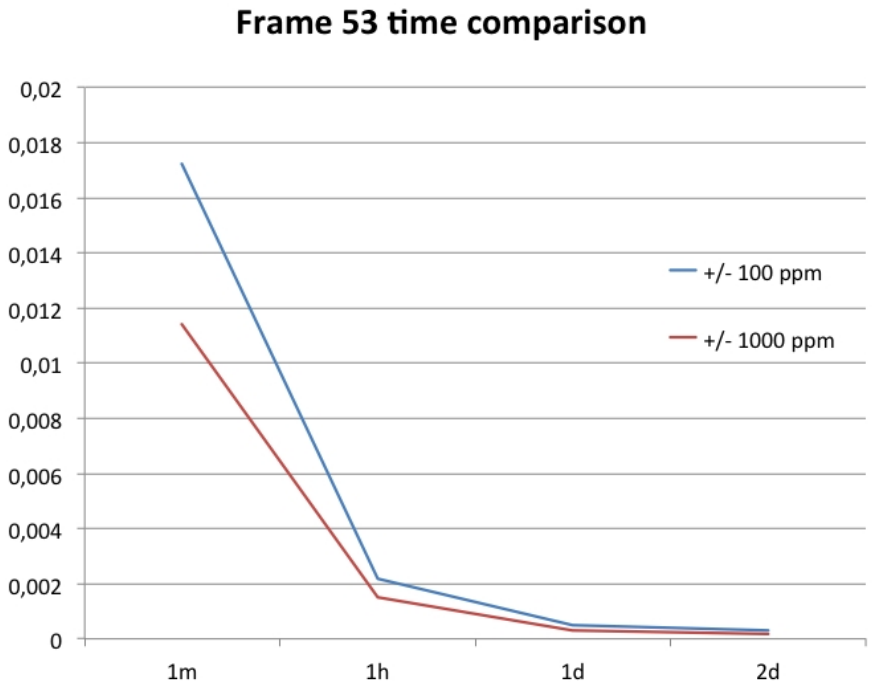

Figure 3.11 – *Evolution temporelle des écart-types des temps de réponses pour 10 simulations avec des déphasages et des dérives aléatoires. La trame étudiée est la trame la moins prioritaire du benchmark SAE. Les durées de temps simulé sont représentées en abscisse et les valeurs moyennes des écart-types sont représentées en ordonnée.*

3.4.5 Comparaison avec l'approche sans dérive d'horloge

Maintenant que nous avons établi que les paramètres initiaux n'influent que faiblement sur les distributions de temps de réponse observés pour de longues observation, il convient de comparer notre approche avec l'approche répandue jusqu'à présent et consistant à réaliser de nombreuses simulations avec des déphasages initiaux différents. En considérant que la granularité minimale pour le choix des déphasages initiaux entre les calculateurs est le temps-bit, et que le PPCM des périodes des trames est 2s, cela donne calculateurs 10^{6N} configurations à simuler sur un réseau à 500 kbit/s avec N. Dans la pratique, il est impossible d'explorer exhaustivement toutes ces possibilités et nous nous limitons donc à nombre plus raisonnable de simulations pour lesquelles les déphasages initiaux des calculateurs sont choisis aléatoirement.

La figure 3.12 montre les distributions de temps de réponse obtenus par les deux

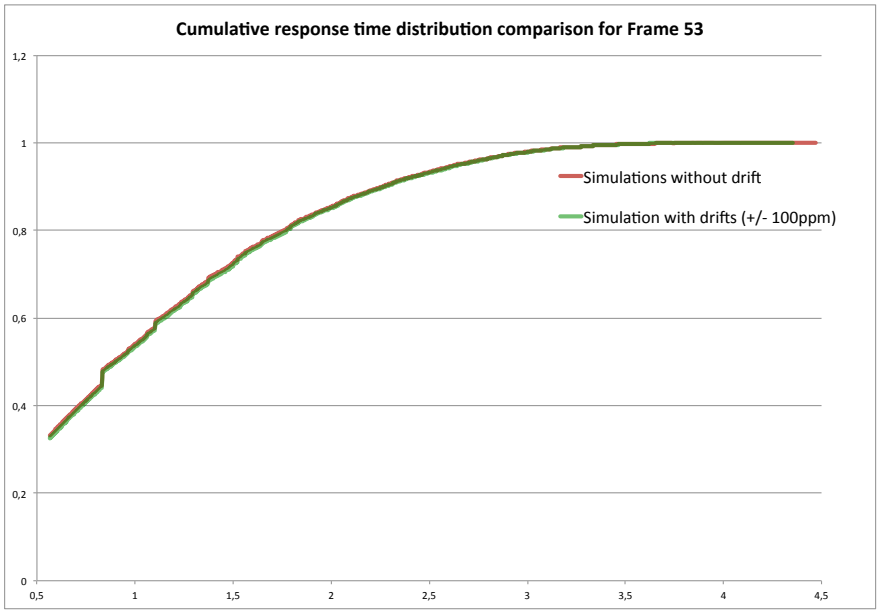

Figure 3.12 – Comparaison des distributions de temps de réponses obtenues par simulations avec et sans dérive d'horloge. La trame étudiée est la trame la moins prioritaire du benchmark SAE. La courbe verte a été obtenue en faisant une seule simulation de 48h de temps simulé avec dérives d'horloges et une borne de +/-100 ppm sur les dérives. La courbe rouge résulte de 100 000 simulations de 20s de temps simulé avec des déphasages initiaux entre calculateurs choisis de manière aléatoire.

différentes méthodes. Pour cela nous nous sommes intéressés spécifiquement à la distribution de temps de réponse de la trame de plus faible priorité du benchmark SAE afin de mieux mettre en évidence les potentiels écarts. La courbe verte donne le résultat obtenus en réalisant une seule simulation de 48h de temps simulé avec une borne de +/-100 ppm sur les dérives d'horloge. La courbe rouge donne la distribution de temps de réponse obtenue en réalisant 100 000 simulations de 20s de temps simulé en choisissant aléatoirement les configurations de départ. La durée de 20s correspond à 20 fois la période du système (dont le PPCM des périodes des trames est 1s) afin de limiter l'impact des temps de réponses mesurés avant que le système ne devienne périodique (du fait du départ décalé des calculateurs).

Le premier constat est que les résultats obtenus sont presques identiques et indique que les deux approches sont équivalentes lorsque l'on cherche à obtenir des distributions de temps de réponse des trames par simulation. En observant de plus près, il est possible d'observer un léger décalage qui prend son origine au meilleur temps de réponse pour lequel il existe une différence de 0,5%. Dans la mesure où la

durée de temps simulé est limitée dans un cas et que le nombre de configurations simulées dans un cas est limité dans l'ordre, il s'agit d'une différence négligeable. Enfin, il est aussi intéressant de noter que les simulations sans dérive d'horloge ont permis dans ce cas d'observer des temps de réponse plus importants que dans l'expérience avec dérive d'horloge. Il s'agit d'une limite de l'approche avec dérives d'horloge pour laquelle les temps de réponses important sont très dépendants de la configuration de départ mais qui est acceptable dans la mesure où ces temps de réponse en question sont très rares et que le pire cas de temps de réponse peut être calculé par analyse.

3.5 Conclusion

Dans ce chapitre, nous avons présenté une approche par simulation pour étudier les réseaux CAN qui tient compte des dérives d'horloge des noeuds communicant sur le bus et qui font varier dans le temps les temps de réponse des trames. Dans la partie 3.3, nous avons pu voir que la simulation et l'analyse ne sont pas suffisantes si on les utilise indépendamment. Tout d'abord, la simulation n'est pas appropriée pour trouver les scénario pire cas de temps de réponse car ces événements sont très rares. Parallèlement, l'analyse pire cas ne donne des indications ni sur la fréquence des scénario pire cas ni sur leur durée. En revanche, en utilisant l'analyse pour trouver les paramètres provoquant un scénario pire cas (spécifique à chaque trame), il est possible de simuler ce scénario afin d'observer combien de temps des grands temps de réponses sont observés et comment se passe la transition induite par les dérives d'horloge. Ces simulations permettent également de valider les résultats obtenus par les outils d'analyse, ce qui est utile car les outils d'analyse sous souvent des boîtes noires et aussi car, même si des progrès sont faits régulièrement [35, 17] des hypothèses simplificatrices sont souvent utilisées par rapport au matériel et à la pile de communication [51]. Les réfléxions résultant de ces études sur des courtes durées de temps simulé ont également débouché sur des dépôts de brevets [7, 8, 9]. Une description de ces inventions est disponible en Annexe C.

Enfin, nous avons établi que les statistiques des temps de réponses observés pendant une simulation finissent par se stabiliser au fur et à mesure que la simulation progresse et que les déphasages initiaux entre calculateurs n'ont pas d'impact sur les distributions finales obtenues. En revanche, nous avons pu observer que les valeurs des dérives d'horloge influencent les résultats. Cependant, le dernier cas d'étude a montré que pour des valeurs typiques de dérive d'horloge, les temps de réponse observés sur de grandes durées de simulation sont les mêmes, quels que soient aussi les déphasages initiaux. Cela confirme donc qu'une seule simulation suffisamment longue est suffisante pour obtenir les statistiques utiles au dimensionnement pour les concepteurs. Cela rend donc pertinent l'étude des distributions de temps de réponse d'une trame dans la mesure où, pour de longues durées de simulations, la

distribution de temps de réponse de chaque trame semble être unique. Ce constat a été confirmé expérimentalement en comparant notre approche avec une approche sans dérive d'horloge telle qu'utilisée habituellement pour obtenir des distributions de temps de réponse. En effet, les résultats obtenus par les deux approches sont identiques.

Dans cette perspective, nous étudions dans le chapitre 4 comment calculer par analyse une estimation des distributions de temps de réponse des trames. En revanche, nous nous ne baserons sur des hypothèses ignorant les dérives d'horloge comme les résultats expérimentaux ont montré l'équivalence des deux approches pour les longues durées de simulation et pour les valeurs de dérives tolérées dans l'industrie automobile.

Remarques

Nous tenons à remercier Rob Davis (Université de York) qui nous a aidé à trouver les paramètres des expériences montrant l'impact de la dérive d'horloge en 3.4.3.

Les résultats de ces travaux ont été publiés dans [45] et [46]. À noter que des expérimentations non présentées ici car réalisées sur des messageries de chez PSA Peugeot-Citroën ont permis de conforter les résultats présentés dans ce chapitre.

Chapitre 4

Évaluation de la distribution de temps de réponse d'une trame CAN

Résumé Ce chapitre s'inscrit dans la continuité du chapitre précédent dans lequel nous avons pu obtenir des distributions de temps de réponses en utilisant une approche par simulation modélisant les dérives d'horloge. Une limitation inhérente aux approches de simulation est la non-exhaustivité des situations étudiées pendant la simulation. Ainsi, pour contourner ce problème, nous cherchons à calculer la distribution de temps de réponse d'une trame CAN via une approche analytique. Pour cela nous empruntons le concept de message caractéristique déjà utilisés dans des travaux similaires par Zeng et al. Ensuite, nous décrivons et expliquons l'allure de distributions de temps de réponse obtenues par simulation et qui comportent de curieux sauts. Sur la base de ces remarques, nous présentons ensuite notre approche d'estimation des distributions de temps de réponse d'une trame CAN. Au regard de la quantité de calculs à effectuer, nous introduisons également des paramètres d'approximations permettant de réduire considérablement le nombre de calculs sans nécessairement perdre en précision de manière remarquable. Enfin, nous comparons expérimentalement les résultats obtenus par analyse et par simulation et discutons les avantages et les inconvénients des deux approches.

Sommaire

4.1 Analyse des distributions des temps de réponse des trames CAN

4.1.1 Objectifs des travaux

Dans ce chapitre, nous cherchons à évaluer les distributions de temps de réponse des trames CAN par analyse. En effet, nous avons établi dans le chapitre 3 que ni les déphasages initiaux entre calculateurs ni la dérive des horloges, pour des valeurs réalistes, n'influençaient les statistiques résultantes de manière significative. En d'autres termes, il est possible de caractériser les performances temporelles d'une messagerie CAN d'une manière unique et pertinente, sur le long terme, quelles que soient les conditions initiales, sous la forme des distributions de temps de réponse des trames composant la messagerie. Nous avons également montré que la méthode utilisée jusqu'à présent pour obtenir des distributions de temps de réponse sans dérive d'horloge donnait des résultats identiques à notre approche avec des dérives d'horloge. Nous explorons ici la possibilité d'obtenir ces distributions de manière analytique comme alternative à la simulation. Nous cherchons à calculer la distribution de temps de réponse d'une trame à partir des seuls paramètres de la messagerie. L'objectif de ce chapitre est de proposer une approche probabiliste pour estimer les distributions de temps de réponses des trames CAN.

4.1.2 Travaux existants

Les premiers travaux probabilistes sur CAN réalisés par Navet et al. [53, 56] s'intéressaient à évaluer la probabilité de respect des échéances des trames à partir d'un modèle d'erreurs suivant un processus de Poisson généralisé. D'autres travaux probabilistes dans le domaine temps-réel se sont ensuite plutôt intéressé à l'ordonnancement de tâches, à commencer par Gardner [21] qui a étudié la probabilité qu'une instance d'une tâche ne respecte pas une échéance. Ensuite, des approches pour calculer les fonctions de densité de probabilité des temps de réponse de tâches périodiques dans les cas mono-coeur et multi-coeurs ont été proposées [20, 38]. De retour aux réseaux CAN, Navet et al. ont étudié la probabilité qu'une trame manque son échéance à cause d'une erreur de transmission dans les scénario pire cas [54]. Par la suite, et toujours dans le contexte du scénario pire cas pour une trame, Nolte et al. [57] ont proposé une approche probabiliste afin de tenir compte du fait que les trames sont de longueurs variables à cause du bit-stuffing. Enfin, Zeng et al. [68] ont proposé une approche stochastique pour le réseaux CAN. Cette dernière approche s'appuie sur les travaux de Diaz et al. [20, 38] pour l'ordonnancement de tâches et est étendue aux réseaux CAN. Pour cela, les auteurs introduisent la notion de message caractéristique pour abstraire le trafic provenant de chaque calculateur communiquant sur le réseau afin de réduire la complexité du problème.

Pour une messagerie de taille moyenne (69 trames), l'approche proposée est capable de calculer assez rapidement des probabilités de temps de réponse proches de celle obtenues par de longues simulations.

4.1.3 Contributions

Dans ce chapitre, nous décrivons une approche pour estimer par analyse les distributions de temps de réponse des trames émises sur un bus CAN. L'évaluation des temps de réponse de chaque trame est réalisée individuellement. Ainsi, tour à tour, chacune des trames de la messagerie devient alors l'objet central des calculs d'estimation de distribution de temps de réponse. L'ordre dans lequel les trames sont étudiées n'a pas d'importance. Cette approche réutilisera des concepts similaires à ceux introduits dans le chapitre II, en particulier le concept de slots et des tables d'ordonnancement lors du calcul des messages caractéristiques. L'approche proposée par Zeng et al. [68] étudiait déjà le calcul des probabilités de temps de réponse des trames émises sur un réseau CAN. Dans nos travaux, nous nous rapprochons de cette approche en nous appuyant en particulier sur le concept de messages caractéristiques qui est une abstraction très puissante pour réduire l'espace des états à explorer lors de l'analyse. Nous commençons donc par expliquer comment calculer ces messages caractéristiques à partir des tables d'activation des trames afin de prendre en compte les offsets entre les trames émises depuis un même calculateur. Nous détaillons ensuite en 4.3.1 des distributions de temps de réponse (obtenues par simulation) et expliquons leur allure, comportant des sauts. Sur la base de ces observations, nous détaillons ensuite notre approche se basant sur l'explication de ces sauts au lieu de modéliser les déphasages entre calculateurs sous la forme de gigue comme fait dans [68]. Nous introduisons également des paramètres permettant d'influencer la complexité des calculs. Enfin, nous comparons expérimentalement notre approche avec les résultats issus de simulation et étudions l'influence des paramètres d'estimation.

4.2 Modèle et notations

4.2.1 Modélisation des trames CAN

Dans ce chapitre, afin de pouvoir comparer les résultats avec ceux obtenus dans le chapitre précédent, nous utilisons les mêmes hypothèses que celles décrites en 3.2.1 . Les trames CAN sont toutes supposées périodiques. Un message CAN m_i est défini par (ID_i, T_i, O_i, C_i) avec ID_i son identifiant CAN (définissant sa priorité), T_i sa période, O_i son offset initial et C_i son temps de transmission. Dans le contexte de ces travaux, nous considérons systématiquement le pire cas de bit-stuffing pour C_i. Ainsi cette valeur dépendra uniquement de la taille des données émises et de la

vitesse du bus.

Les offsets utilisés ici ont le même emploi que ceux utilisés pour les runnables périodiques dans le chapitre 2. Ils permettent de désynchroniser les émissions des trames envoyées par un même noeud afin de réduire les situations de surcharge temporaire et réduire les temps de réponse. Cela est désormais une pratique courante dans les réseaux de communication automobiles (voir par exemple [24]). Nous supposons donc ici qu'un outil comme Netcar-Analyzer [61] a été préalablement utilisé pour calculer les valeurs des offsets des trames.

Pour les calculs réalisés pour l'analyse des temps de réponse, nous négligerons alors l'effet de la dérive car celui-ci n'est pas sensible au regard des valeurs étudiées. En effet, si on considère que le temps de réponse typique d'une trame sur un réseau à 500 kbit/s est de l'ordre de 1ms, alors une dérive de 150ppm entraîne une déviation de 0,15 μs, ce qui est négligeable pour un temps de réponses compris entre 100 μs et plusieurs millisecondes. Nous avons également vu expérimentalement dans le chapitre précédent que les résultats obtenus avec dérives d'horloge et ceux obtenus sans dérive d'horloge mais avec des déphasages initiaux différents étaient identiques. Ainsi, dans ce chapitre, nous baserons nos calculs sur l'approche traditionnellement utilisée où dérives d'horloge sont nulles et les configurations de déphasages initiaux entre les calculateurs sont supposées équiprobables.

4.2.2 Modélisation de l'activation des trames

Au niveau local de chaque calculateur, nous cherchons à modéliser l'activation de messages périodiques. L'activation d'une trame est le moment où celle-ci est prête à participer au processus d'arbitration pour accéder au bus CAN. Les trames considérées ici sont périodiques et nous connaissons leurs déphasages relatifs lorsqu'elles sont émises par la même station. Ainsi l'activation des trames est similaire à de l'ordonnancement statique cyclique et donc à l'ordonnancement des runnables étudié dans le chapitre II. Pour cette raison, nous pouvons utiliser un modèle similaire à celui des tâches séquenceurs du chapitre 2.

Pour chaque calculateur, nous pouvons ainsi décrire l'activation des trames CAN via une table d'ordonnancement décomposée en "slots". Chaque slot contient les trames à émettre suite à l'activation de la tâche de communication au niveau du middleware (cf Figure 3.2). Ainsi la période entre deux slots est la période de cette tâche et correspond aussi à la granularité utilisée pour les offsets des messages. Le placement des trames dans les slots est une conséquence directe du choix de leur offset initial et de leur période et permet d'exprimer les dates d'activation des différents trames. La construction des tables d'activation est similaire aux résultats présentés dans le chapitre 2.

A chaque ECU t communiquant sur le bus CAN étudié, nous associons une table d'activation Ω_t. Un calculateur est donc décrit ainsi : $ECU_t = (T_{COM}, T_{ECU_t}, \Omega_t)$

avec :

- T_{COM} la durée élémentaire d'un "slot", égale à la période de la tâche middleware responsable de la communication
- T_{ECU_t} la période globale de la table d'ordonnancement correspondant à l'ECU t
- Ω_t est la table d'activation des trames émises par ce calculateur

Nous notons $\{m\}_{\Omega_t,k}$ l'ensemble des trames contenus dans le slot k de la table d'activation Ω_i.

Nous faisons l'hypothèse que la durée T_{COM} est égale pour tous les calculateurs communiquant sur un même bus CAN. De manière similaire à l'ordonnancement des runnables, ces valeurs sont typiquement $T_{ECU_t} = 1000\,ms$ et $T_{COM} = 5\,ms$. Et de même, T_{ECU_t} est un multiple du PPCM des périodes des trames tandis que PPCM des périodes des trames doit être un multiple de T_{COM}. En conséquence, il y a T_{ECU_i}/T_{COM} slots dans la table d'activation $\Omega_t(\{m\})$. L'ensemble des trames $\{m\}_{\Omega_t,k}$ contenues dans le slot k de la table d'activation Ω_t seront simultanément prêtes à participer au processus d'arbitration. Elles seront ensuite envoyées chacune à leur tour en fonction de leur identifiant qui déterminent leur priorité. La Figure 4.1 donne un exemple de table d'activation pour un calculateur envoyant 5 trames.

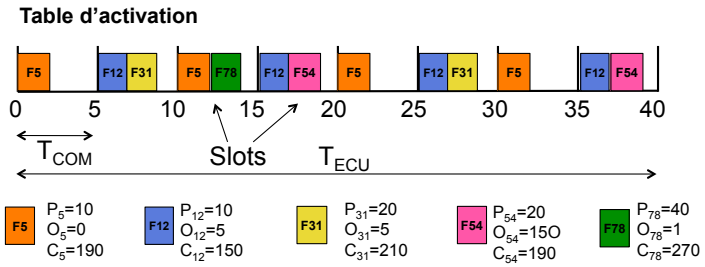

Figure 4.1 – Modèle de table d'activation pour un calculateur émettant 5 trames dont les identifiants respectifs sont 5, 12, 31, 54 et 78. Les temps de transmissions sont exprimés en μs.

4.2.3 Messages caractéristiques

Calculer des distributions de temps de réponse des trames nécessite, pour chaque trame étudiée de calculer et combiner les probabilités des temps de réponse correspondants à un très grand nombre de scénarios de déphasages. Afin de diminuer la complexité du problème, nous présentons ici la notion de message caractéristique telle qu'introduite par Zeng et al. dans [69]. Cette abstraction permet de simplifier le modèle d'activation des trames émises par un ECU sous forme probabiliste. Le but est de représenter, sous la forme d'un message dit "caractéristique",

les interférences des trames plus prioritaires que la trame étudiée de la manière la plus simple possible. Nous distinguons ensuite le cas du calculateur émettant la trame étudiée et les autres calculateurs.

Concept de message caractéristique

Supposons que nous nous intéressons au message m_i émis par le calculateur ECU_{m_i}. Dans la mesure où les calculateurs différents de ECU_{m_i} ne sont pas synchronisés avec celui-ci, une tentative directe d'estimer la distribution de temps de réponse d'une trame nécessiterait de considérer tous les déphasages possibles entre les tables d'activations de tous les calculateurs. Le message caractéristique modélise, sous la forme d'une fonction de masse discrète $\mathscr{M}_{ECU_s}^{m_i}$, les interférences provenant de l'activation des trames plus prioritaires que m_i émises par un calculateur ECU_s différent de ECU_{m_i}. Cela permet de simplifier la table d'activation des calculateurs et de réduire grandement les calculs effectués pour l'analyse.

À la différence de Zeng qui supposait que les offsets des trames émises par un même calculateur étaient nuls (départ simultané), nous nous appuyons sur les tables d'activation des ECUs afin de prendre en compte les offsets des trames entre elles. Afin de calculer $\mathscr{M}_{ECU_s}^{m_i}$, nous calculons d'abord pour chaque slot k de la table d'activation de ECU_s la somme des interférences des messages plus prioritaires que m_i sous la forme d'une fonction de probabilité. Enfin, $\mathscr{M}_{ECU_s}^{m_i}$ est obtenu en faisant la somme de ces fonctions et en divisant par T_{ECUs}/T_{COM} pour normaliser la fonction (pour que la somme de ces valeurs discrètes soit égale à 1). Le principe reste identique et le résultat final a la même forme, mais notre algorithme parcourt la table d'activation au lieu de s'appuyer uniquement sur les périodes des messages. Le calcul de $\mathscr{M}_{ECU_s}^{m_i}$ est décrit par l'algorithme 4.1. La fonction ainsi obtenue permet donc de connaître la probabilité des différentes interférences provenant de ECU_{m_i} à chaque fois qu'un slot est activé pendant que le message m_i est en attente d'être émis. L'algorithme parcourt la table d'activation pour calculer les valeurs b_k, temps de blocage du slot d'indice k, en filtrant les trames moins prioritaires. Chaque valeur participe avec une probabilité égale à $\frac{T_{COM}}{T_{ECU_s}}$ dans le message caractéristique calculé. Ainsi, une fois la table d'activation construite à partir des offsets des trames, le calcul du message caractéristique est rapide (proportionnel au nombre de trame émises par ECU_s).

La Figure 4.2 donne en exemple le résultat correspondant au cas de la table d'activation présentée en Figure 4.1 pour une trame dont l'identifiant est compris entre 32 et 54, donc cette trame est moins prioritaire que la trame d'identifiant 32 et plus prioritaire que la trame d'identifiant 54.. Par ce moyen, au lieu de considérer 8 slots différents, nous n'avons plus que 3 cas à considérer. Dans la pratique, cela permet de réduire généralement à moins de 10 cas distincts des tables d'activations contenant 200 slots. Ceci est d'autant plus efficace pour les trames de plus haute

Algorithme 4.1 Calcul de $\mathscr{M}_{ECU_s}^{m_i}$

for $k = 1, 2, ..., T_{ECUs}/T_{COM}$ **do**
  $b_k = 0$
  for each m_j in $\{m\}_{\Omega_s,k}$ **do**
    if $ID_j < ID_i$ **then**
      $b_k = b_k + C_j$
    end if
  end for
  $\mathscr{M}_{ECU_s}^{m_i}(b_k) = \mathscr{M}_{ECU_s}^{m_i}(b_k) + \frac{T_{COM}}{T_{ECU_s}}$
end for

priorité pour lesquelles le nombre de trames plus prioritaires est réduit. De plus, cela permet de réduire les interférences provenant du nombre potentiellement important de messages émis sur le réseau au nombre d'ECUs. En revanche, transformer les tables d'activation en message caractéristiques fait perdre les informations de précédence entre les slots. Par exemple, en revenant sur l'exemple de la Figure 4.2, une interférence de 400 μs sera toujours suivie par une interférence de 150 μs. Cette information est perdue quand on regarde juste le message caractéristique. Cependant, comme nous le verrons par la suite, prendre en compte l'interférence d'un seul slot par ECU permet d'obtenir la quasi totalité de la distribution de temps de réponse d'une trame. Il est relativement aisé de calculer d'autres messages caractéristiques modélisant les interférences de plusieurs slots consécutifs mais cela nécessiterait de distinguer un grand nombre de possibilités supplémentaires dans l'analyse présentée après, alors qu'elle explore déjà un nombre très importants de cas.

Interférences locales

Le cas du calculateur qui émet la trame est traité de manière similaire. Ainsi, pour chacun des slots de la table d'activation de l'ECU émetteur où la trame émise est présente, les temps de transmission des trames plus prioritaires sont additionnés et exprimés sous la forme d'une fonction de masse notée $\mathscr{M}_{ECU_{m_i}}^{*}$. Cela permet aussi de réduire plusieurs dizaines d'occurrence de la trame à un nombre réduit de valeurs pondérées par des probabilités. La probabilité de la valeur "0" sera donc la probabilité que la trame étudiée soit prête à être envoyée sans attendre qu'une trame provenant du même ECU soit émise en même temps.

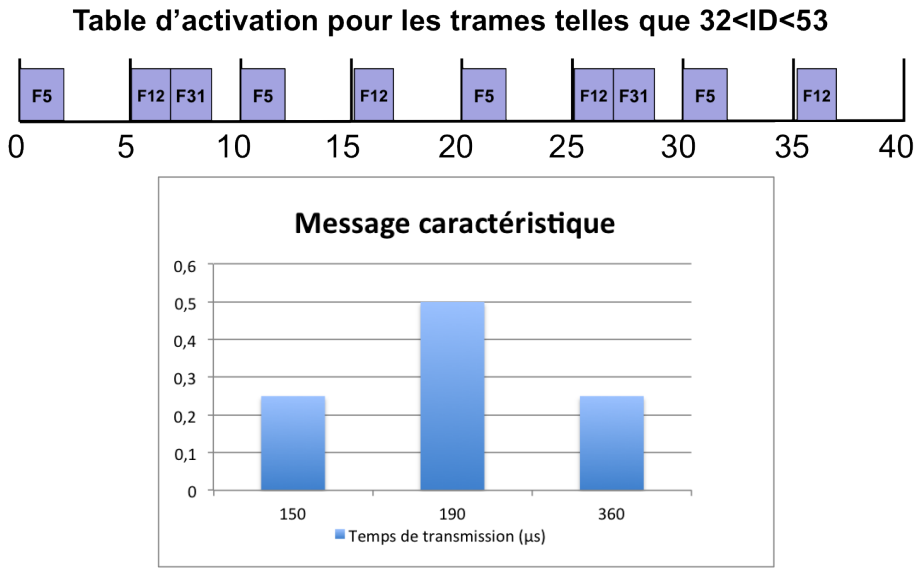

Figure 4.2 – Message caractéristique correspondant à la table d'activation de la figure 4.1 pour un niveau de priorité entre 32 et 53. L'axe des abscisses correspondant à la durée de l'interférence en μs et l'axe des ordonnées à la probabilité associée à celle-ci. Une interférence de $360\mu s$ correspond à l'interférence causée par l'arrivée simultanée des trames F12 et F31 occupant le bus respectivement pendant $150\mu s$ et $210\mu s$.

4.2.4 Visualisation des résultats sous forme de fonction de répartition des temps de réponse

Dans ce chapitre, nous cherchons à calculer des distributions de temps de réponse de trame. Nous notons alors $\mathbb{R}_i(t)$ la fonction de probabilité des temps de réponse de la trame i. $\mathbb{R}_i(t_0)$ donne la probabilité que la trame i ait pour temps de réponse la valeur t_0. Par nature de l'ordonnancement sur un réseau CAN, $\mathbb{R}_i(t)$ est une fonction discrète de granularité τ, le temps bit du bus CAN (c'est-à-dire le temps de transmission d'un bit de donnée). En effet, les valeurs des temps de réponses sont multiples de τ car il s'agit de la durée du plus petit événement pouvant arriver sur le réseau CAN (la transmission d'un bit).

Afin de représenter graphiquement les distributions de temps de réponse, nous utiliserons de manière privilégiée la fonction de répartition correspondante $\mathscr{F}_i(t) = \mathbb{R}_i([0,t])$. Dans la mesure où les probabilités de temps de réponses deviennent très faibles pour les grands temps de réponse, la fonction de répartition permet de mieux visualiser et comparer les résultats. Afin de comparer les résultats obtenus par analyse avec les statistiques obtenues par simulation, nous utiliserons les courbes

de statistiques cumulées représentant en chaque point la somme de la statistique de temps de réponse de ce point et des statistiques constatées avant.

La figure 4.3 donne un exemple d'une distribution de temps de réponse pour une trame CAN et de la fonction de répartition associée pour une trame de priorité moyenne.

4.3 Prévision stochastique des distributions des temps de réponse des trames

Maintenant que nous avons présenté les moyens de modéliser les interférences provenant des trames émises par un calculateur sous la forme d'un seul objet probabiliste, nous décrivons notre approche. Tout d'abord, nous étudions des résultats issus de simulation afin de mettre en évidence des "points d'accumulation" qui serviront de motivation à l'approche proposée. Ensuite, après avoir présenté la structure générale de l'approche, nous détaillons les différentes phases de celle-ci en décrivant les différents cas possibles avant d'expliquer comment en faire la synthèse. A cette occasion, nous décrivons et étudions comment paramétrer la finesse de l'approche afin de trouver un compromis entre la précision des résultats et la quantité de cas à considérer lors des calculs.

4.3.1 Analyse de résultats de simulation : existence de sauts dans les distributions de temps de réponse

Dans un premier temps, nous observons deux fonctions de répartition empiriques obtenues par simulation avec dérive d'horloge d'un bus CAN dérivé d'une application réelle. Le réseau est composé de 13 ECUS émettant un ensemble de 68 trames à un débit de 500 kbit/s et générant une charge d'environ 30% sur le bus. Les courbes présentées ici donnent les statistiques cumulées enregistrées en utilisant l'approche présentée dans le chapitre 3 après stabilisation des courbes. La granularité de l'axe horizontal est le temps-bit et chaque point correspond donc à un temps de réponse précis et non à un intervalle de temps de réponse. L'axe verticale donne la somme des fréquences des temps de réponses inférieures à la valeur en abscisse correspondante.

Cas des trames prioritaires La première courbe en Figure 4.4 donne les statistiques cumulées des temps de réponse de la quatrième trame la plus prioritaire de la messagerie. Elle est très bien représentative des statistiques de temps de réponses observées pour les trames les plus prioritaires. Pour une valeure de temps de réponse donnée, la probabilité associée est la probabilité de transmettre une trame en une durée inférieure ou égale à ce temps de réponse. A son extrême gauche se trouve

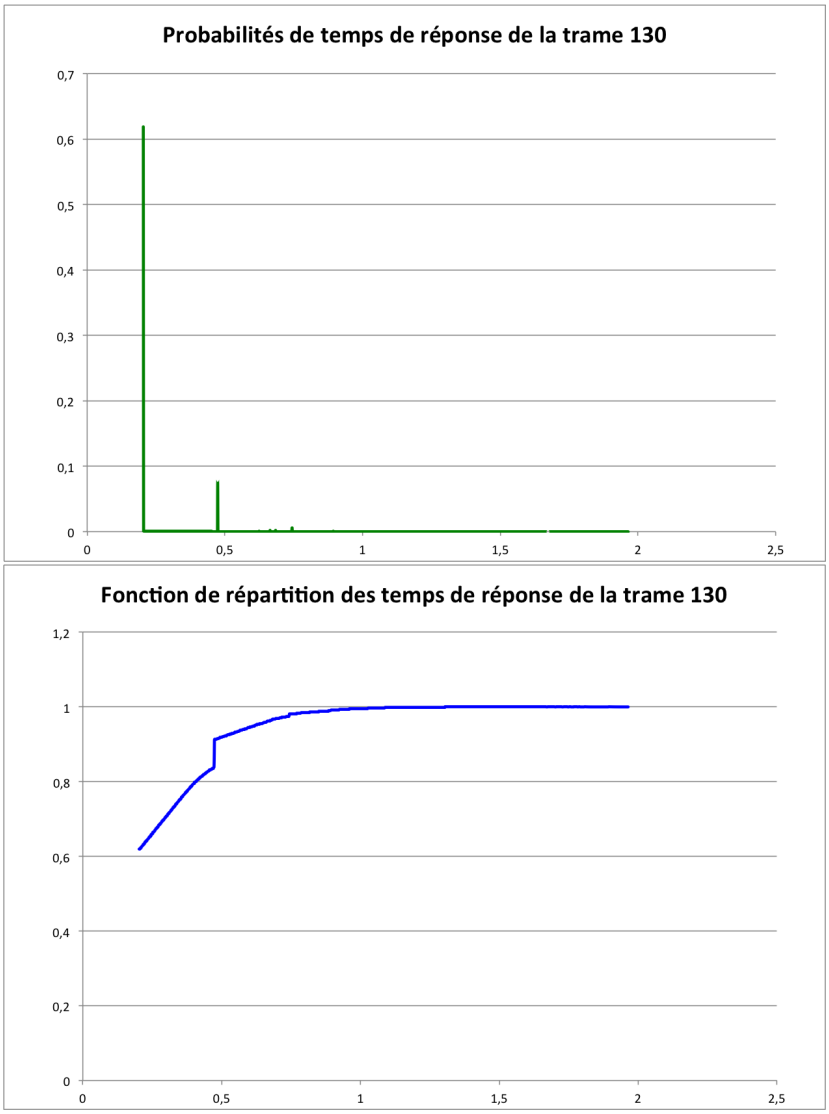

Figure 4.3 – Probabilité de temps de réponse et fonction de répartition correspondante pour la trame d'identifiant 130 issue d'une messagerie PSA. Nous utiliserons la seconde représentation à partir de maintenant pour des questions de lisibilité car l'écart entre la valeur maximum et les autres valeurs rend la première représentation peu utilisable.

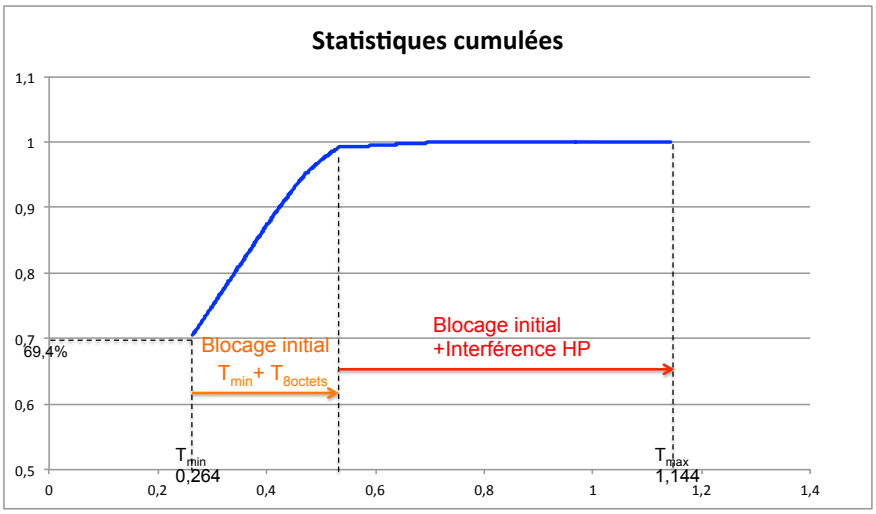

Figure 4.4 – Statistiques cumulées des temps de réponse d'une trame de haute priorité (4ème plus prioritaire sur 68) sur un réseau avec 13 ECUs à 500 kbit/s chargé à 30%

le meilleur temps de réponse observé avec une très grande fréquence, ici 69,4%. Il s'agit du cas de sa transmission directe dans le cas où le bus est libre au moment où elle commence à participer au processus d'arbitrage. Nous avons donc ici un temps de réponse de 264 μs correspondant au temps de transmission d'une trame avec 8 octets de données utiles à 500 kbit/s dans le pire cas de bit-stuffing. Sa fréquence correspond à peu près à "1-charge du bus", ce qui est cohérent. La suite de la courbe se décompose en deux parties : une première phase avec une pente moyenne et une deuxième phase à l'approche de la valeur 1 de type plateau, avec une pente très faible. La première partie correspond aux cas où le bus est occupé au moment où la trame est prête à être transmise. Cette phase a une largeur correspondant au temps de transmission d'une trame de taille maximale (8 octets de données utiles). La dernière partie correspond aux cas rares (d'où la faible pente), où une (ou plusieurs) des trame(s) plus prioritaire(s) que la trame correspondante devient prête à être émise pendant le blocage initial du bus. Le temps de réponse maximum mesuré ici est 1,144 ms avec une fréquence égale à 5.10^{-7}.

Cas des trames de priorité plus faible Cependant, en progressant vers les trames de priorités plus faibles, des courbes avec des caractéristiques différentes émergent. Ainsi, les statistiques cumulées pour les trames moins prioritaires sont plutôt semblables à l'exemple présenté dans la Figure 4.5 obtenu avec la vingtième trame dans l'ordre de priorité de la messagerie utilisée. En plus des caractéristiques

observées précédemment : une fréquence importante de la valeur la plus faible, une section croissante immédiatement après et un plateau à la fin, nous pouvons clairement distinguer des sauts verticaux comme par exemple entre 500 μs et 800 μs et plusieurs fois ensuite de manière plus discrète. Ces sauts indiquent que certaines valeurs de temps de réponses particulières sont observées très fréquemment par rapport aux autres. Ces points d'accumulation sont très visibles lorsque l'on regarde les fonctions de probabilités comme par exemple dans la Figure 4.3, aux dépends des autres valeurs qui sont très faibles en comparaison.

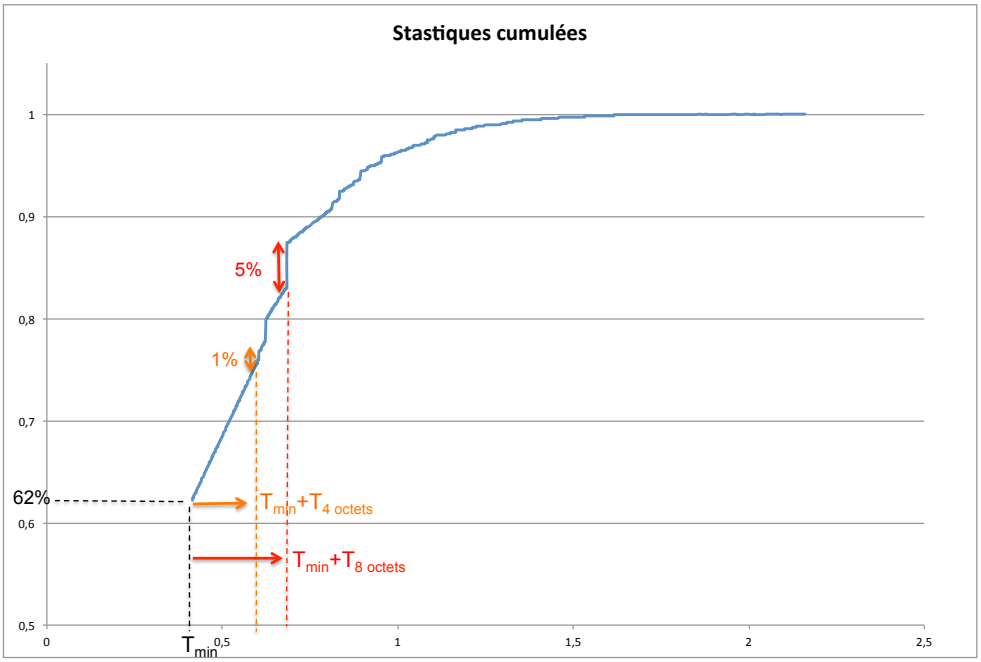

Figure 4.5 – Statistiques cumulées des temps de réponse d'une trame de priorité moyenne (20ème plus prioritaire sur 68) sur un réseau avec 13 ECUs à 500 kbit/s chargé à 30%.

Pour mieux comprendre ce qui se passe, regardons tout d'abord le meilleur temps de réponse de la trame. Il est égal à 414 μs alors que son temps de transmission est égal à 224 μs (6 octets de données utiles à transmettre à 500 kbit/s avec le pire cas de bit-stuffing). Une analyse des offsets révèle en réalité que la trame étudiée est systématiquement envoyée en même temps qu'une trame plus prioritaire comportant 4 octets de données utiles et qui induit donc un temps d'attente de 190 μs (184 pour transmettre la trame et 3 τ_{bit}=6μs d'attente entre chaque trame). Pendant ce délai systématique, si une trame plus prioritaire que la trame étudiée est prête à être émise, elle retardera d'autant plus la transmission de la

trame étudiée. Ce qui rend ce phénomène remarquable est que, pendant le temps d'attente systématique, quel que soit l'instant précis où une trame plus prioritaire devient prête à être émise, la trame étudiée devra attendre la transmission de cette nouvelle trame. Ainsi, quand on étudie des trames qui sont régulièrement retardées car émises en même temps que d'autres trames plus prioritaires (elles sont dans le même slot de la table d'activation), ces sauts montrant l'accumulation de temps de réponse aux niveaux de certaines valeurs particulières va pouvoir être observé. Cette accumulation est dû à la combinaison de deux faits. Tout d'abord, l'intervalle pendant lequel des trames plus prioritaires peuvent arriver est relativement important car l'instant exact d'arrivée des trames plus prioritaires est indifférent pendant le temps d'attente initial et la probabilité qu'aucune autre trame n'arrive en plus reste important aussi (dans le cadre d'une charge de réseau moyenne typique, de 30% à 60% en fonction des constructeurs). Le deuxième fait est une conséquence des hypothèses de travail qui font que les temps de transmission des trames ne dépendent que de la taille des données utiles. Ainsi les sauts se situent à des endroits précis correspondant à la somme du temps de réponse minimale et du temps de transmission de message comportant un certain nombre d'octets de données utiles. Sur la figure, nous pouvons ainsi distinguer un premier saut correspondant à l'arrivée de trames plus prioritaires avec 4 octets de donnée pendant le temps de blocage initial. Le saut le plus important correspondant à 5% des temps de réponse correspond à l'arrivée de trames plus prioritaires comportant 8 octets de données. D'autres sauts sont également visibles sur la courbe. Ils correspondent aux interférences de plusieurs trames plus prioritaires, phénomène plus rare et donc marqué par de plus petites ruptures de pente.

En réalité, comme les longueurs des trames peuvent varier un peu à cause du bit-stuffing, les sauts ne seraient pas aussi nets dans la pratique. Les mêmes sauts pourraient être observés sur quelques temps de réponses rapprochés car les statistiques de temps de réponses seraient en partie redistribuées vers des temps légèrement plus courts dans la mesure où l'on n'observe pas toujours le pire cas de bit-stuffing en pratique.

L'objectif de ces travaux est de calculer, uniquement à partir des paramètres de la messagerie CAN, une fonction de répartition proche des statistiques observées par simulation. Pour faire face à la complexité de l'analyse, nous nous appuyons sur trois idées. La première idée consiste à utiliser les messages caractéristiques pour simplifier grandement le modèle. La seconde idée consiste à utiliser les observations que nous venons de faire et qui indiquent que pour qu'une trame plus prioritaire augmente le temps d'attente de la trame étudiée, l'instant exact d'arrivée pendant le temps d'attente déjà accumulé est indifférent. Enfin, nous introduisons des paramètres permettant de maîtriser la profondeur de l'analyse. En effet, certaines combinaisons d'événements très rares ne contribuent que très faiblement au résultat qui se comporte rapidement de manière asymptotique.

4.3.2 Présentation de l'approche

Pour le calcul de la distribution de temps de réponse d'une trame m_i émise par ECU_{m_i}, l'étape difficile est de calculer la distribution de ses temps d'attente avant qu'elle puisse être transmise. Le passage de la distribution de temps d'attente à la distribution de temps de réponse se fait simplement en ajoutant le temps de transmission de la trame dans la mesure où il n'y a pas de possibilité de préemption lors de l'envoi d'une trame sur un bus CAN. La dernière étape consistant à transformer la fonction de probabilité en sa fonction de répartition pour la présentation des résultats est également relativement directe.

Nous proposons une approche en deux phases. La première phase de cette approche consiste à évaluer le temps d'attente initial de la trame m_i sous forme probabiliste. Cette étape nécessite de considérer conjointement l'état du réseau au moment où la trame est prête à être émise, comme détaillé en 4.3.3 ainsi que la présence éventuelle de trame plus prioritaires dans les slots contenant la trame étudiée dans la table d'activation Ω_t de ECU_{m_i} qui l'émet comme expliqué en 4.2.3. La deuxième phase consiste à explorer les scénarios d'arrivée de trames plus prioritaires suite à un blocage initial comme décrit en 4.3.5. Pour cela nous calculons d'abord les possibilités d'arrivées de trames plus prioritaires sous la forme d'un "arbre d'interférences". Ensuite, pour chacune des valeurs de blocage initial, nous parcourons l'arbre d'interférences et calculons les probabilités de chaque chemin ainsi que la valeur totale des temps d'attente en utilisant les messages caractéristiques calculés préalablement.

Dans la mesure où la taille de l'arbre d'interférence augmente très rapidement, nous utilisons deux paramètres pour maîtriser l'exploration. L'utilisation de ces paramètres ainsi que d'autres hypothèses de travail nous induisent à négliger certains cas rares et surestimer la probabilité de certains temps de réponse faibles. La distribution de temps de réponse obtenue est donc légèrement optimiste par rapport à celles obtenus par simulation.

Une conséquence sous-jacente de l'utilisation des messages caractéristiques est l'introduction d'erreur si le temps d'attente dépasse la valeur T_{COM} impliquant le début d'un deuxième slot des tables d'activation participant au blocage de la trame étudiée car les relations de précédences entre les slots sont perdues. Ainsi, dans la suite de l'approche, nous faisons le choix de ne considérer l'interférence d'un seul slot par message caractéristique afin de simplifier les calculs. Les erreurs introduites par cette hypothèse ne se font sentir que pour des temps de réponses proches et supérieures à T_{COM}. Dans la pratique, ces temps de réponses sont dans les queues de distribution et participe de manière infinitésimale à la partie asymptotique des courbes des fonction de répartition. Cette hypothèse est généralement sans conséquence négative pour les trames qui ne sont pas de très basse priorité et/ou si la charge du bus n'est pas trop élevée (i.e., environ 30% dans les cas étudiés).

4.3.3 Occupation du bus au moment de l'instantiation d'une trame

Dans cette première étape de l'analyse, nous regardons les différentes possibilités au niveau de ce qui se passe sur le bus au moment où la trame étudiée est prête à être émise. Nous distinguons quatre cas :

- le bus est occupé par une trame émise par le même calculateur,
- le bus est libre,
- le bus est occupé par une trame moins prioritaire émise par un autre calculateur,
- le bus est occupé par une trame plus prioritaire émise par un autre calculateur.

Bus occupé par une trame émise par le même calculateur

Dans notre approche ce cas est négligé. Ce cas survient si une trame présente dans un slot de la table d'activation Ω_t précédant un slot contenant m_i a un temps de réponse supérieur à la longueur d'un slot. Dans le contexte de l'utilisation de méthode de calcul d'offsets réduisant grandement les pires temps de réponse, cette situation est rare. Il est facile de vérifier que ce cas n'arrive pas en calculant le pire temps de réponse des trames contenus dans les slots précédents ceux contenant la trame étudiée. Si aucune trame n'a un pire temps de réponse que la durée d'un slot T_{COM}, alors cela n'arrive jamais. Dans le cas où cela arrive, il est possible de constater via simulation par exemple la rareté de ces cas généralement situés dans la partie asymptotique des statistiques cumulées (ou de la fonction de répartition si le résultat est obtenu par analyse).

Bus libre

Le cas suivant est le cas où le bus est libre. Il s'agit d'un cas assez fréquent et qui induira le meilleur temps de réponse de la trame. Dans le contexte où l'on néglige les interférences provenant du calculateur émettant la trame, la probabilité que le bus soit libre est obtenue en retirant la charge induite par les trames émises par les autres calculateurs. Le temps de blocage initial pour ce cas est donc nul et sa probabilité est donnée par l'équation 4.1.

$$p_{bus\,libre}(m_i) = 1 - \sum_{\{m_j \setminus m_j \text{émise par } ECU_s \neq ECU_{m_i}\}} \frac{C_j}{T_j} \tag{4.1}$$

Blocage par une trame moins prioritaire émise par un autre calculateur

Le cas suivant est celui où le bus est occupé par l'émission d'une trame de plus basse priorité émise par un autre calculateur. Dans notre approche nous faisons l'hypothèse d'un "backlog" nul au moment où m_i est prête à être émise, c'est à

dire qu'aucune trame de plus haute priorité est en attente d'être émise en plus de la trame qui occupe actuellement le bus. Dans le cas où la trame m_j en train d'être émise est de plus faible priorité que m_i alors le seul scénario où le backlog serait non-nul est le cas où une trame plus prioritaire que m_i devient prête à être transmise entre le moment où la trame sur le bus a débuté sa transmission et l'arrivée de m_i (si elle était arrivée avant alors elle aurait été envoyée à la place de m_j). Ceci peut s'estimer en considérant toutes les combinaisons d'une trame moins prioritaire émise par un autre calculateur qu'ECU_{m_i} ainsi que d'une trame plus prioritaire provenant encore d'un autre calculateur. Dans la pratique, cela dépend de la charge du réseau et de la position de la trame dans la liste de priorité. Cette hypothèse est rarement contredite pour les trames les plus prioritaires (comme le nombre de trames plus prioritaires pouvant arriver est faible). Cela pose également peu de problèmes pour les trames les moins prioritaires pour lesquelles la probabilité que le bus soit occupé par une trame moins prioritaire est faible et le cas où cette hypothèse n'est pas vérifiée d'autant plus rare.

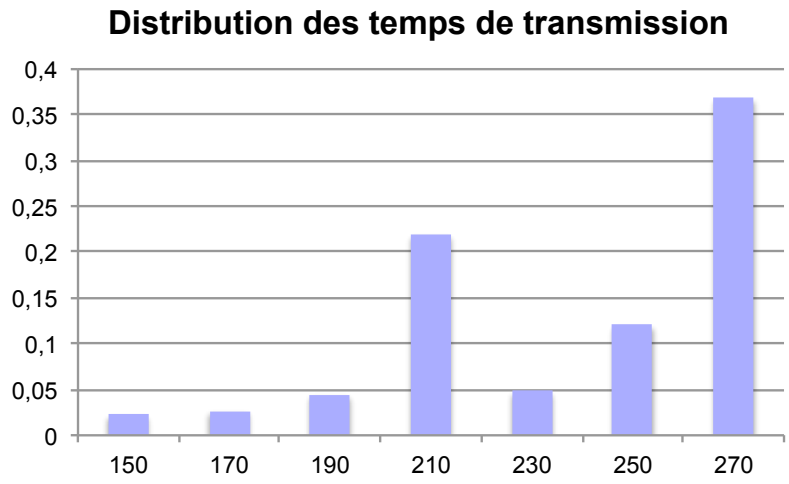

Figure 4.6 – Exemple de $\mathcal{T}_{bp}^{m_i}$, distribution des temps de transmission des trames moins prioritaires émises par les autre calculateurs que ECU_{m_i} pour une trame de haute priorité issue d'une messagerie didactique utilisée chez PSA Peugeot-Citroën sur un bus à 500 kbit/s. Il n'y a que sept valeurs car il n'y a pas de trames moins prioritaires avec un seul octet de données utiles, ce qui correspondrait à un temps de transmission de $130\mu s$.

Nous cherchons donc à calculer la distribution des temps de blocage du bus par les trames moins prioritaires que m_i émises par d'autres calculateurs que ECU_{m_i}. Cela se fait en deux étapes. La première étape consiste à calculer la fonction de

probabilité des temps d'occupation du bus pour les trames les moins prioritaires. Nous calculons donc la distribution des temps de transmission des trames de plus basse priorité $\mathscr{T}_{bp}^{m_i}$. En itérant sur les trames m_j telle que $ID_j > ID_i$, on ajoute la probabilité d'occupation de cette trame du bus à la probabilité du temps de transmission correspondant (initialisée à 0) selon la formule 4.2. La Figure 4.6 donne un exemple d'une telle distribution.

$$\mathscr{T}_{bp}^{m_i}(C_j) = \mathscr{T}_{bp}^{m_i}(C_j) + \frac{C_j}{T_j} \tag{4.2}$$

L'étape suivante consiste à calculer la distribution $\mathscr{B}_{bp}^{m_i}$ des temps de blocage induits par les trames moins prioritaire. Nous faisons l'hypothèse que les instants possibles d'arrivée de la trame étudiée pendant l'occupation du bus par une autre trame sont équiprobables en accord avec l'hypothèse selon laquelle les déphasages entre calculateurs sont équiprobables. Ainsi, si la trame occupant le bus comporte 2 octets de données utiles, on considère de manière égale les possibilités de blocage entre $\tau_{bit} = 2\mu s$ et 150 μs. Par saut de τ_{bit} qui est la granularité des événements considérés, nous faisons alors la synthèse $\mathscr{B}_{bp}^{m_i*}(t)$ des trames qui sont susceptibles de bloquer pendant t unités de temps entre 1 τ_{bit} et la valeur maximale de temps de transmission (135 τ_{bit} s'il y a des trames avec 8 octets de données utiles) en additionnant les valeurs $\mathscr{T}_{bp}^{m_i}(C_k)$, pour tous les $C_k \geq t$, ce qui s'écrit comme suit.

$$\mathscr{B}_{bp}^{m_i*}(t) = \sum_{C_k \geq t} \mathscr{T}_{bp}^{m_i}(C_k) \tag{4.3}$$

Enfin $\mathscr{B}_{bp}^{m_i}$ est obtenu en normalisant $\mathscr{B}_{bp}^{m_i*}$ de manière à ce que la somme de ses valeurs soit $p_{bp}(m_i) = \sum \mathscr{T}_{bp}^{m_i}(C_k)$, la probabilité que le bus soit occupé par une trame de plus basse priorité. La Figure 4.7 présente la distribution $\mathscr{B}_{bp}^{m_i}$ correspondant à la Figure 4.6.

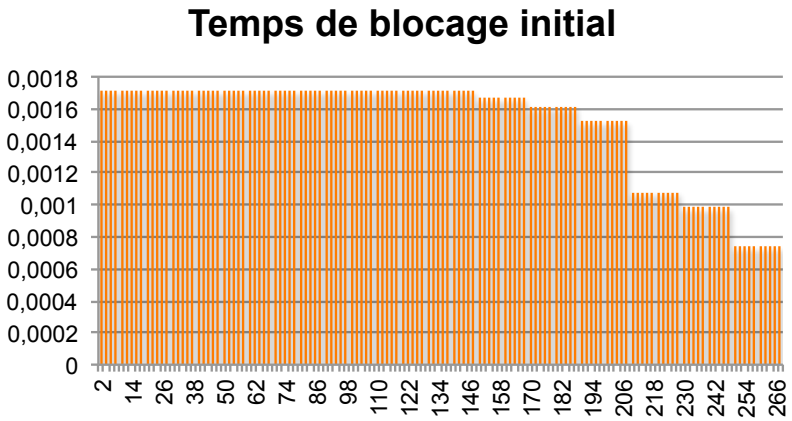

Figure 4.7 – Exemple de $\mathscr{B}_{bp}^{m_i}$, distribution du blocage initial du bus par une trame de plus basse priorité que m_i émise par un autre calculateur que ECU_{m_i} correspondant à la distribution de temps de blocage de la Figure 4.6.

Distribution du blocage résultant de l'occupation du bus par des trames plus prioritaires La dernier cas est celui où le bus est bloqué par une trame de plus haute priorité que la trame étudiée. Contrairement aux trames de plus basses priorité qui ne peuvent pas retarder m_i au-delà de l'occupation initiale du bus, il est important de tracer l'origine du trafic de plus haute priorité qui peut venir allonger le temps d'attente de m_i. Ce choix de considérer séparément les trames plus prioritaires lors de l'étude de l'état du bus vise à éviter de compter deux fois l'interférence des trames plus prioritaires : une fois durant cette étape, et une seconde fois dans la seconde phase de l'approche qui explore mes scénario d'interférence des trames de plus haute priorité.

En ce qui concerne le backlog, nous faisons aussi la même hypothèse que le cas précédent et considérons qu'il n'y avait pas de trames plus prioritaire en attente d'être émise. Dans le cas où une trame m_j occupant le bus est plus prioritaire que m_i, cela devient plus compliqué que le cas précédent. Soit une trame plus prioritaire que m_j peut arriver pendant l'émission de m_j soit des trames de priorités comprises entre celles de m_i et m_j étaient déjà présentes dans la file d'attente. En pratique, ces cas sont rares et peuvent être quantifiés par simulation. Prendre en compte ces cas demanderait de faire des calculs similaires aux nombreux calculs effectués par la suite. Dans le contexte de ces travaux, nous avons décidé de les négliger.

Pour calculer les interférences des trames plus prioritaires que m_i émises par un calculateur ECU_s différent de ECU_{m_i}, nous procédons de la même manière que pour les trames moins prioritaires mais en utilisant les messages caractéristiques. Soit $\mathscr{T}_{hp,ECU_s}^{m_i}$ la distribution de probabilité des temps d'occupation du bus des trames

plus prioritaire que m_i et émises par ECU_k. Ses valeurs sont obtenues en multipliant les valeurs du message caractéristique $\mathscr{M}^{m_i}_{ECU_s}$ par leur taux d'occupation du bus donné par leur temps d'interférence divisé par T_{COM}, la période des slots des tables d'activation. Soient $\{b_k\}$ les temps de blocage définies pour $\mathscr{M}^{m_i}_{ECU_s}$, nous avons la relation suivante :

$$\mathscr{T}^{m_i}_{hp,ECU_s}(b_k) = \mathscr{M}^{m_i}_{ECU_s}(b_k) \cdot \frac{b_k}{T_{COM}} \tag{4.4}$$

Les fonctions $\mathscr{B}^{m_i*}_{hp,ECU_s}$ puis $\mathscr{B}^{m_i}_{hp,ECU_s}$ sont obtenues en procédant comme dans le cas des trames moins prioritaires à partir de $\mathscr{T}^{m_i}_{hp,ECU_s}$.

4.3.4 Interférences locales

Au blocage initial du bus vient s'ajouter éventuellement un temps d'attente supplémentaire induit par la présence de trames plus prioritaires présentes dans les mêmes slots que m_i dans la table d'activation d'ECU_{m_i}. Ce temps de blocage supplémentaire est décrit via le message caractéristique local $\mathscr{M}^{*}_{ECU_{m_i}}$ comme expliqué en 4.2.3. Les combinaisons de ces temps de blocage local et des scénarios de blocage de bus permet de décrire les différents scénarios de blocage initial pendant lesquels des trames plus prioritaires provenant d'autres stations peuvent arriver. L'arrivée ultérieure d'interférences de plus haute priorité doit à présent être étudiée.

4.3.5 Interférences des autres stations

La deuxième phase consiste à explorer les scénarios possibles d'interférences provoquées par les arrivées de trames plus prioritaires que m_i envoyés par d'autres calculateurs qu'ECU_{m_i} pendant que m_i est en attente d'être émise sur le réseau. Pour cela, nous nous reposons sur les observations issues des exemples décrits en 4.3.1. Dans cette dernière phase de l'approche, nous avons deux choses à faire. Dans un premier temps nous calculons l'"arbre d'interférences" de m_i qui décrit tous les cas possibles d'interférences des ECUs différents d'ECU_{m_i} qui émettent au moins une trame plus prioritaire. Ensuite, nous étudions chacun des scénarios d'interférences correspondant aux chemins depuis la racine vers chacune des feuilles de l'arbre. Chacun de ces scénarios est interprété en utilisant les messages caractéristiques et les situations de blocage initiales calculées précédemment afin d'obtenir une distribution de temps d'attente pour m_i. La synthèse de toutes les distributions obtenues donne la distribution de temps d'attente de m_i.

Arbre d'interférences

Afin d'énumérer tous les scénarios décrivant les interférences des trames plus prioritaires pendant le temps d'attente de la trame étudiée m_i, nous calculons un "arbre d'interférences". Nous avons observé en 4.3.1 que le moment exact d'activation d'une trame plus prioritaire provenant d'un autre calculateur n'est pas important tant que cet événement se produit pendant que la trame émise est en attente que le bus se libère. Pour cette raison, nous énumérons tous les scénarios possibles sans se soucier de la valeur des temps d'attentes en se concentrant juste sur le déroulement ou non des événements correspondant à l'activation d'un slot d'un calculateur émettant des trames plus prioritaires. Chacun de ces scénarios permet alors potentiellement de capturer un grand nombre de configurations de déphasages entre les calculateurs.

Si on considère les calculateurs différents de ECU_{m_i} qui émettent au moins une trame plus prioritaire que m_i, il y a possibilité d'interférence si et seulement si un slot de la table d'activation d'un de ces ECUs débute pendant le temps d'attente de m_i. Dans le cas où une interférence se produit, cela induit un temps d'attente supplémentaire pendant lequel de nouvelles interférences peuvent alors se produire. Cela se répète jusqu'à ce que tous les calculateurs aient participé aux interférences ou dans le cas ou aucune nouvelle interférence ne se produit pendant le temps d'attente supplémentaire ce qui permet alors l'émission de la trame étudiée. L'énumération de ces scénarios nécessite uniquement de connaître quels sont les ECUs qui émettent des trames plus prioritaires que m_i.

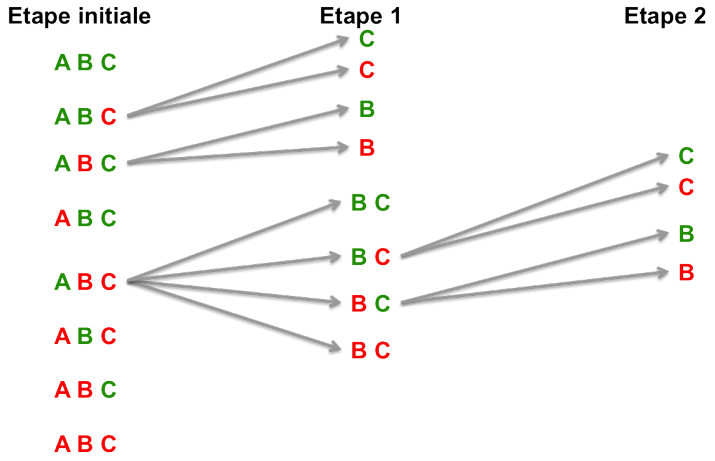

Figure 4.8 – Exemple d'"arbre d'interférences" pour 3 calculateurs A, B et C de plus haute priorité. Le vert indique l'interférence d'une trame provenant du calculateur correspondant. Le rouge signifie qu'aucun slot n'a débuté pendant le temps d'attente de l'étape en cours.

Afin de comprendre comment procéder, prenons en exemple le cas où trois calculateurs A, B et C émettent au moins une trame plus prioritaire que m_i. La Figure 4.8 énumère les scénarios possibles. La premier niveau de l'arbre, l'étape initiale" sur la gauche de l'arbre décrit l'ensemble des possibilités pendant les temps de blocages calculés dans $f_{blocage\,initial}$. Le vert signifie qu'un slot de la table d'activation du calculateur correspondant a débuté pendant le temps d'attente. Le rouge représente l'événement complémentaire où aucun slot n'a débuté pendant le temps d'attente considéré.

Soit k le nombre de calculateurs qui "interfèrent" parmi n. Il y a donc $\sum_{k=0}^{n} \binom{n}{k}$ alternatives, ici 8 possibilités pour 3 calculateurs. Considérons d'abord les cas extrêmes. Si aucun calculateur n'interfère pendant le temps de blocage initial, alors la trame m_i est émise. Si tous les calculateurs interfèrent pendant le blocage initial, alors nous pouvons nous arrêter également. En effet, comme expliqué précédemment, nous nous limitons dans notre approche à l'interférence d'un slot par calculateur et une fois que tous les calculateurs ont participé, il suffit alors d'ajouter la somme des interférences induites données par leurs messages caractéristiques avant d'être en mesure de transmettre la trame. En revanche, il est nécessaire de continuer l'exploration dans les cas intermédiaires. Par exemple, si A et B interfèrent et non C pendant le temps de blocage initial, alors pendant le temps de blocage supplémentaire induit par les trames plus prioritaires émises par A et B, il faut alors considérer le cas où C interfère et le cas où C n'interfère pas. De manière similaire si seulement A arrive, alors il faut refaire cette étude pour les calculateurs B et C pour le temps de blocage supplémentaire induit par l'arrivée d'une trame plus prioritaire émise par A. En pratique, pour chaque chemin où 2 calculateurs n'interfèrent pas, il faut recalculer les possibilités d'interférences possibles pour ces 2 calculateurs pendant le nouveau temps de blocage ce qui revient à refaire le calcul de départ pour 2 calculateurs au lieu de 3. Ainsi, intuitivement, si l'on note $taillearbre(n)$, le nombre de chemins possibles, cette valeur peut être calculée itérativement par la formule 4.5. L'ajout de "1" correspond au cas où tous les calculateurs interfèrent. Ensuite, pour toutes les possibilités où k calculateurs parmi n n'arrivent pas, il faut recompter le nombre de chemins pour (n-k) calculateurs.

$$\begin{cases} taillearbre(0) = 1 & n = 0 \\ taillearbre(n) = 1 + \sum_{k=1}^{n} \binom{n}{k} \cdot taillearbre(n-k) & n > 0 \end{cases} \tag{4.5}$$

La profondeur de l'arbre d'interférence pour n calculateurs est n. Cette profondeur est atteinte sur les chemins où un seul calculateur interfère à chaque étape. En revanche, la taille de cet arbre explose rapidement (environ 200 millions de branches pour 10 calculateurs). Cependant, le contenu de l'arbre ne dépend que du nombre de calculateurs. Une fois les chemins calculés pour n calculateurs donnés, il suffit de remplacer les calculateurs utilisés dans l'arbre par de nouveaux lors d'un nouveau cas faisant intervenir n calculateurs différents. Dans la pratique, nous interprétons

chacun des scénarios au-fur-et-à-mesure qu'ils sont construits afin de ne pas avoir besoin de garder en mémoire l'ensemble de l'arbre ce qui devient impossible à partir d'un certain nombre de calculateurs.

Interprétation des scénarios d'interférences

Chacun des chemins obtenus en parcourant l'arbre depuis la racine vers les feuilles correspond à un scénario possible d'interférence. Le dernier maillon nécessaire pour mener à bien notre approche consiste à interpréter ces scénarios sous la forme de distributions de temps de réponse.

Chacun des scénarios est décomposé en plusieurs étapes correspondant à chacun des noeuds de l'arbre sur le chemin parcouru. Chacune des étapes décrit quels calculateurs interfèrent et quels calculateurs n'interfèrent pas (représentés respectivement en vert et rouge dans la Figure 4.8. Chaque scénario commence par une étape initiale (étape 1) correspondant à ce qu'il se passe pendant un temps de blocage initial. Les temps de blocage initiaux résultent de la combinaison de l'état du bus (cf 4.3.3) et des interférences locales (cf 4.3.4). Soit b_{init} un temps de blocage initial, $p_{b_{init}}$, la probabilité associée à ce temps de blocage, $n_{interf}(1)$ le nombre de calculateur interférant à l'étape initiale (l'étape 1), $n_{non-interf}(1)$ le nombre de calculateur émettant des trames plus prioritaires n'interférant pas pendant cette étape initiale, alors $p_1(b_{init})$, la probabilité de cette étape étape initiale suite à un temps de blocage de durée b_{init}, est donnée par l'équation 4.6.

$$p_1(b_{init}) = (\frac{b_{init}}{T_{COM}})^{n_{interf}(0)} \cdot (1 - \frac{b_{init}}{T_{COM}})^{n_{non-interf}(0)} \tag{4.6}$$

Étant donné un temps d'attente b_{init} et le fait que les calculateurs débutent un slot avec une période T_{COM}, la probabilité qu'un calculateur entame un slot pendant ce temps est $\frac{b_{init}}{T_{COM}}$.

Lors de l'interprétation d'un scénario d'interférence, nous notons $\mathscr{I}_k$ la fonction de probabilité des nouvelles interférences après le déroulement de l'étape k. Pour représenter le blocage initial, nous définissons $\mathscr{I}_0$ sous la forme d'une seule valeur : $\mathscr{I}_0(b_{init}) = p_{b_{init}}$. Enfin, nous notons $\{ECU\}_{interf,k}$ l'ensemble des calculateurs interférant à l'étape k, de cardinal $n_{interf}(k)$ et par extension $n_{non-interf}(k)$ le nombre de calculateurs n'interférant pas à l'étape k. Comme il n'y a qu'un seul temps de blocage initial, $\mathscr{I}_1$ est obtenue en faisant la convolution de leurs messages caractéristiques comme suit :

$$\mathscr{I}_1 = p_1(b_{init}) \cdot \mathscr{I}_0(b_{init}) \bigotimes_{ECU_s \in \{ECU\}_{interf,1}} \mathscr{M}_{ECU_s}^{m_i} \tag{4.7}$$

Pour les formules à venir, nous notons $\{F\}$ l'ensemble des valeurs sur lesquelles une fonction de probabilité F est définie. À l'étape suivante du scénario étudié, il

faut procéder de même pour chacun des nouveaux temps de blocage $\{\mathscr{I}_1\}$ pour lesquels $\mathscr{I}_1$ est définie à la place de b_{init} et ainsi de suite pour chacune des étapes du scénario d'interférences étudié. À chaque nouvelle étape, nous utilisons l'interférence supplémentaire induite pendant l'étape précédente pour calculer la probabilité d'interférer et de ne pas interférer pour les calculateurs qui ne l'ont pas fait jusqu'alors. Pour cela, nous définissons $\mathscr{I}_k^*$ une fonction de probabilité définie pour des doublets $(b_{total}, b_{courant})$ représentant respectivement la somme des interférences subies jusqu'à l'étape k et le nouveau temps de blocage correspondant aux nouvelles interférences arrivées pendant l'étape précédent l'étape k. Soit l le nombre d'étapes pour le scénario étudié, l'algorithme 4.2 détaille les calculs effectués pour calculer $\mathscr{I}_{s_{interférence}, b_{init}, p_{b_{init}}}$, la distribution de temps de réponse correspondant à ce scénario d'interférence après un blocage initial b_{init} de probabilité $p_{b_{init}}$

Algorithme 4.2 Calcul de $\mathscr{I}_{s_{interférence}, b_{init}, p_{b_{init}}}$

$\mathscr{I}_0^*((b_{init}, b_{init})) = p_{b_{init}}$
for k=1...l **do**
 for each $(b_{total}, b_{courant})$ **in** $\{\mathscr{I}_{k-1}^*\}$ **do**
  $p_k(b_{courant}) = (\frac{b_{courant}}{T_{COM}})^{n_{interf}(k)} \cdot (1 - \frac{b_{courant}}{T_{COM}})^{n_{non-interf}(k)}$
  $\mathscr{M}_k^{m_i} = \bigotimes_{ECU_s \in \{ECU\}_{interf,1}} \mathscr{M}_{ECU_s}^{m_i}$
  for each $b_{interference}$ **in** $\{\mathscr{M}_k^{m_i}\}$ **do**
   $\mathscr{I}_k^*((b_{total} + b_{interférence}, b_{interférence})) = p_k(b_{courant}) \cdot \mathscr{I}_{k-1}^*((b_{total}, b_{courant})) \cdot \mathscr{M}_k^{m_i}(b_{interférence})$
  end for
 end for
end for
$\mathscr{I}_{s_{interférence}, b_{init}, p_{b_{init}}} = \mathscr{I}_l$

Ainsi, un partir d'un temps de blocage initial et un scénario d'interférence, nous obtenons une partie de la distribution de temps d'attente de la trame m_i du fait de l'introduction des messages caractéristiques. Cela signifie aussi que pour chacun des nombreux chemins de l'arbre d'interférence, nous ajoutons encore de la complexité dans l'approche. Le nombre de temps d'attentes résultant dépend du nombre de messages caractéristiques et de leur nombre de valeurs. Dans le pire cas, il peut être nécessaire de calculer un nombre de temps d'attente égale au produit de la taille des messages caractéristiques.

4.3.6 Synthèse de l'approche

Pour être capable de mettre en oeuvre l'approche présentée ici, il reste encore à expliquer comment les différentes opérations de calculs s'enchaînent les unes

avec les autres. En particulier, nous n'avons pas encore expliqué comment passer de la phase de calcul des blocages initiaux à la phase où l'arbre d'interférence est simultanément construit est interprété. En pratique, nous explorons plusieurs arbres d'interférences en fonction de la nature du blocage initial. En effet, la raison pour laquelle nous distinguons le blocage du bus selon que les trames sont de plus faibles ou plus hautes priorités que la tramé étudiée est d'éviter de compter deux fois le blocage d'une trame plus prioritaire. En conséquence, lorsque nous étudions ce qui peut se passer suite au blocage initial du bus par une trame émise par un calculateur ECU_s différent de ECU_{m_i}, nous calculons et interprétons l'arbre d'interférence en enlevant ECU_s à la liste des calculateurs intervenant dans les scénarios d'interférence. Ainsi, en partant de la description de la messagerie, le calcul de la fonction de répartition des temps de réponse d'une trame m_i émise par ECU_{m_i} requiert d'effectuer les opérations qui suivent.

Calculs préparatoires

Dans un premier temps, il est nécessaire de transformer la messagerie sous une forme plus pratique pour effectuer des calculs. Dans cette phase de préparation, nous commençons par construire les tables d'activation des calculateurs en respectant les propriétés des trames. Une fois les tables d'activation construites pour chacun des calculateurs, nous calculons alors les messages caractéristiques $\mathscr{M}^{m_i}_{ECU_s}$ pour tous les calculateurs différents de ECU_{m_i} et enfin $\mathscr{M}^{*}_{ECU_{m_i}}$ pour ce dernier. Enfin, nous finissons cette phase préparatoire en étudiant l'état du bus au moment de l'instantiation de m_i en calculant les valeurs de $p_{bus\ libre}(m_i)$, $\mathscr{B}^{m_i}_{bp}$ et les différents $\mathscr{B}^{m_i}_{hp,ECU_s}$.

Calcul de la distribution des temps d'attente de m_i

À présent, il faut construire les arbres d'interférences et interpréter leurs scénarios pour chacune des situations de blocage initiales possibles. Les situations de blocages initiales sont le résultat de la combinaison de l'état du bus et du message caractéristique local. Nous notons A_{m_i} l'arbre d'interférence calculé pour tous les calculateurs émettant des trames plus prioritaires que m_i et $A_{m_i,E\bar{C}U_s}$ l'arbre d'interférence pour tous les calculateurs émettant des trames plus prioritaires que m_i sauf ECU_s. Le calcul de $\mathscr{I}_{m_i}$, la distribution des temps d'attente de m_i est détaillé par l'algorithme 4.3.

Ainsi l'arbre A_{m_i} est interprété pour chaque blocage initial résultant de la combinaison d'une valeur de blocage local b_{loc} et du blocage du bus par une trame moins prioritaire b_{bp}. Il est aussi d'abord interprété aussi une fois pour chaque valeur de b_{loc} pour le cas où le bus est libre. De manière similaire, les arbres $A_{m_i,E\bar{C}U_s}$ sont interprétés pour chaque combinaison d'une valeur de blocage local b_{loc} et de b_{hp},

Algorithme 4.3 Calcul de $\mathscr{I}_{m_i}$ la distribution des temps d'attente de m_i

calculer A_{m_i}
for each b_{loc} **in** $\mathscr{M}^*_{ECU_{m_i}}$
$\mathscr{I}_{m_i} = \mathscr{I}_{m_i} + \sum_{s_{interférence} \in A_{m_i}} \mathscr{I}_{s_{interférence}, b_{loc}, p_{bus\ libre}(m_i)}$
for each b_{bp} **in** $\mathscr{B}^{m_i}_{bp}$
$b_{init} = b_{loc} + b_{bp}$
$p_{b_{init}} = \mathscr{M}^*_{ECU_{m_i}}(b_{loc}) + \mathscr{B}^{m_i}_{bp}(b_{bp})$
$\mathscr{I}_{m_i} = \mathscr{I}_{m_i} + \sum_{s_{interférence} \in A_{m_i}} \mathscr{I}_{s_{interférence}, b_{init}, p_{b_{init}}}$
end for
for each $ECU_s \neq ECU_{m_i}$
calculer $A_{m_i, E\bar{C}U_s}$
for each b_{hp} **in** $\mathscr{B}^{m_i}_{hp, ECU_s}$
$b_{init} = b_{loc} + b_{hp}$
$p_{b_{init}} = \mathscr{M}^*_{ECU_{m_i}}(b_{loc}) + \mathscr{B}^{m_i}_{hp, ECU_s}(b_{hp})$
$\mathscr{I}_{m_i} = \mathscr{I}_{m_i} + \sum_{s_{interférence} \in A_{m_i, E\bar{C}U_s}} \mathscr{I}_{s_{interférence}, b_{init}, p_{b_{init}}}$
end for
end for
end for

le blocage initial du bus induit par les trames plus prioritaires contenues dans un slot de la table d'activation de ECU_s.

Calcul de la fonction de répartition des temps de réponse de m_i

La fonction de répartition des temps de réponse de m_i est obtenue à partir de la fonction de distribution des temps de réponse de m_i, notée $\mathscr{R}_{m_i}$et calculée en ajoutant le temps de transmission C_i de la trame m_i à chacune des valeurs pour laquelle $\mathscr{I}_{m_i}$est définie.

$$\forall t_{attente} \in \{\mathscr{I}_{m_i}\},\ \mathscr{R}_{m_i}(t_{attente} + C_i) = \mathscr{I}_{m_i}(t_{attente}) \tag{4.8}$$

4.3.7 Paramètres d'approximation

Dans la pratique, interpréter un arbre d'interférence demande de faire beaucoup de calculs. A fortiori, interpréter les arbres d'interférences pour toutes les valeurs possibles de blocage initial peut demander un nombre impressionnant de calculs

(comme nous le verrons ensuite). De fait, nous introduisons trois paramètres d'approximation permettant de diminuer le nombre de calcul à effectuer pour obtenir les temps de réponse pour les messageries mettant à l'épreuve notre approche. L'influence de ces paramètres sera étudiée expérimentalement dans la section suivante.

Granularité du blocage initial du bus

Par défaut, nous utilisons τ_{bit} comme granularité pour les calculs dans la mesure où il s'agit de la durée de l'événement le plus élémentaire considéré dans cette analyse : la transmission d'un bit de donnée sur le bus. Afin de gagner en temps de calcul, il est possible d'utiliser des multiples de τ_{bit} pour limiter le nombre de cas à explorer par la suite lors des calculs de $\mathscr{B}_{bp}^{m_i}$ et $\mathscr{B}_{hp,ECU_s}^{m_i}$ détaillés en 4.3.3. Utiliser $k \cdot \tau_{bit}$ comme granularité permet de réaliser k fois moins de calculs. En conséquence, la granularité de la distribution de temps de réponse calculée sera $k \cdot \tau_{bit}$. Un bon choix est 5 τ_{bit} dans la mesure où une trame avec un octet de données utiles est transmis en 65 τ_{bit} et chaque octets supplémentaire nécessite 10 τ_{bit} de plus (8 bits à transmettre + 2 pour le pire cas de bit-stuffing). Par ailleurs, c'est le choix qui a été effectué dans [68]. Ainsi, cela permet toujours de capturer séparément les différents d'une fonction de répartition de temps de réponse et l'allure résultat obtenu est donc identique à la solution par défaut à l'exception de la granularité des valeurs de temps de réponse.

Seuil de probabilité

Un deuxième moyen de limiter le nombre de calculs est de fixer un seuil de probabilité P_{min} et d'écarter les cas pour lesquels la probabilité est inférieure à ce seuil lors de l'interprétation des scénarios d'interférence. En effet, il n'est pas forcément intéressant de continuer à interpréter un scénario d'interférence si les probabilités calculées à partir de l'étape courante sont extrêmement faibles. Choisir $P_{min} = 10^{-9}$ par exemple permet de diminuer le nombre de calcul sans perdre beaucoup en précision.

Seuil de profondeur de l'arbre d'interférence

Le dernier moyen de simplifier les calculs est de fixer une borne d_{max} pour la profondeur des arbres d'interférences. Ainsi, lors du calcul des chemins d'interférence, on limite le nombre d'étapes sur un chemin à cette valeur. Cela peut permettre de limiter la taille des arbres d'interférences. En revanche, utiliser cette approximation ajoute un biais à l'analyse. En effet, un chemin limité à d_{max} étapes ajoutera un nombre plus faible de temps de réponse et avec des probabilités plus grandes par rapport au chemin complet qui se serait ramifié en plusieurs chemins parmi lesquels il y aurait des chemins où les ECUs qui n'ont pas encore interféré auraient

pu le faire. Interpréter un chemin qui a été coupé a pour conséquence de compter certains temps de réponse avec des probabilités plus grandes par rapport à l'arbre complet pour lequel une partie des probabilités de ces temps de réponses aurait été reportée pour des temps de réponses plus important (correspondant aux scénarios d'interférences manquant).

4.4 Résultats expérimentaux

L'approche présentée dans ce chapitre a été implémentée en java. Les calculs sont facilement parallélisables et notre implémentation supporte le multithreading ce qui apporte un gain de performance non négligeable pour ces calculs intensifs sur les machines modernes possédant généralement plusieurs coeurs logiques. Dans un premier temps, nous comparons les résultats obtenus par simulation et par analyse. Nous étudions ensuite l'influence des paramètres d'approximation sur la précision et la longueur des calculs. Enfin, nous discutons les avantages et les inconvénients de l'approche par analyse par rapport à l'approche par simulation.

4.4.1 Comparaison avec la simulation

Pour cette série d'expérimentations, nous utiliserons deux messageries. La première messagerie est un benchmark didactique de chez PSA Peugeot Citroën, inspiré de messageries utilisées en production. La configuration étudiée décrit les échanges de 67 trames entre 13 calculateurs sur un bus à 500 kbit/s chargé à 33%. La seconde messagerie est celle utilisée dans [68] qui décrit les échanges de 69 trames entre 6 calculateurs sur un bus à 500 bit/s chargé à 60%. Pour chacune des messageries nous avons calculé les offsets entre les trames émises depuis le même ECU avec l'algorithme "Similar Offsets" de Netcar-Analyzer [61]. Pour chaque trame choisie, nous calculons la fonction de répartition de ses temps de réponse avec l'approche présentée dans ce chapitre et nous la comparons avec celle obtenue par simulation. Les résultats de simulation sont obtenus pour 48h de temps simulé avec des dérives d'horloges limitées à 100 ppm. Des simulations sans dérives d'horloge à partir d'un grand nombre de configurations aléatoires de déphasages entre calculateurs ont été effectuées mais nous ne les représentons pas ici car elles se superposent à celles réalisées avec dérives d'horloge.

Dans un premier temps, nous cherchons à valider notre approche sur quelques exemples de trames issues du benchmark didactique de PSA Peugeot Citroën. Les Figures 4.9, 4.10 et 4.11 montrent respectivement les résultats obtenus pour les trame d'identifiant 112, 158 et 163.

La première trame, étudiée dans la Figure 4.9, possède une fonction de répartition de temps de réponse sans sauts. Les courbes obtenues par les deux approches sont très proches malgré de très légers écarts au niveau de la cassure de la pente.

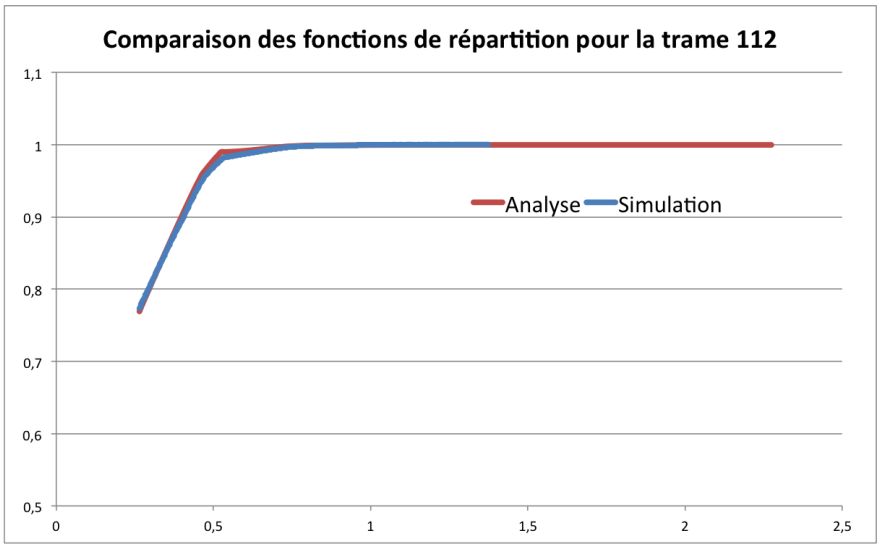

Figure 4.9 – Comparaison des fonctions de distributions pour la trame 112 (11ème dans l'ordre de priorité sur 68) du benchmark didactique de PSA obtenues par analyse (en rouge) et par simulation (en bleu). La courbe obtenue par analyse est très proche de celle obtenue par simulation avec de légers écarts pour les valeurs avec une ordonnée comprise entre 95% et 98%.

Cela s'explique par le fait que l'hypothèse selon laquelle il n'y a pas de trame plus prioritaire en attente au moment ou la trame 112 est prête à être émise n'est pas toujours respectée. En revanche, l'analyse a été capable de capturer plus de temps de réponses. Ceci était attendu car les simulations ne balaient pas de manière exhaustive tous les scénarios de déphasages possibles. En conséquence, il est difficile de capturer les temps de réponses les plus importants qui sont extrêmement rares en pratique.

La seconde trame, étudiée dans la Figure 4.10, possède une fonction de répartition avec des sauts. La courbe obtenue par analyse reproduit fidèlement les sauts observés par simulation. Cette fois aussi, de légers écarts sont présents avant d'atteindre la partie asymptotique des courbes. L'analyse est également allée plus loin dans le calcul des temps de réponse pour les mêmes raisons que pour la trame précédente.

La Figure 4.11 montre les résultats obtenus pour la trame 163. Même si les deux courbes sont assez proches, l'écart est cette fois-ci plus marqué. L'endroit où l'écart est le plus important correspond aux derniers temps de réponses correspondant à l'envoi de la trame après le blocage initial du bus. Le résultat obtenu par analyse surestime la probabilité des temps de réponse correspondant à ces scénarios. Ce phénomène est donc clairement lié à l'hypothèse sur la nullité du backlog

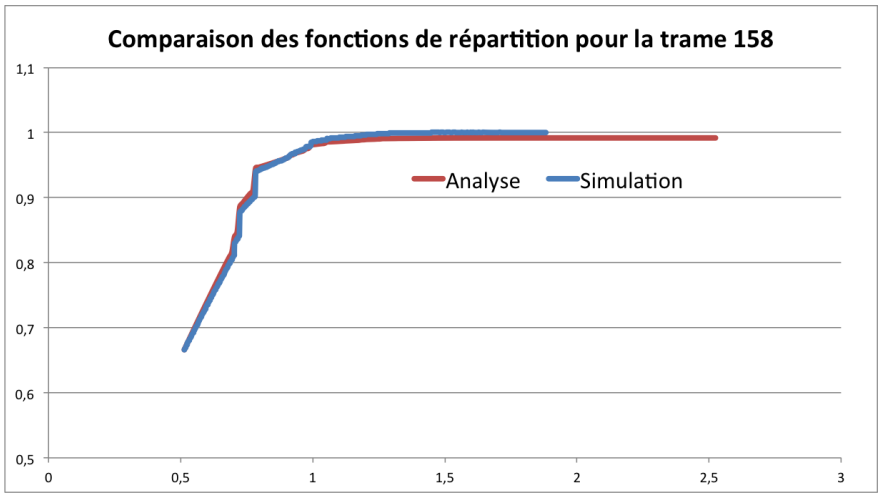

Figure 4.10 – Comparaison des fonctions de distributions pour la trame 158 (15ème dans l'ordre de priorité sur 68) du benchmark didactique de PSA obtenues par analyse (en rouge) et par simulation (en bleu). L'approche par analyse a été capable de retrouver très fidèlement les sauts observées par simulation.

selon laquelle aucune trame plus prioritaire est en attente d'être émise quand la trame étudiée commence à participer au processus d'arbitration. L'erreur due à cette hypothèse va en s'intensifiant au fur et à mesure que la charge plus prioritaire augmente (donc pour des trames de priorités plus faibles). Pour observer ce phénomène, nous utilisons donc dans la suite le benchmark issu de [68] pour lequel la charge réseau est nettement plus importante pour un nombre similaire de trames car celles-ci ont des périodes plus faibles.

La Figure 4.12 correspond à une trame de priorité moyenne. Cette messagerie induit une charge supérieure sur le bus CAN ce qui met à mal nos hypothèses de travail. Comme attendu, nous observons donc des écarts entre les courbes obtenues par les deux différentes méthodes. Malgré cela, les courbes sont très proches et laissent envisager l'utilisation de notre approche pour obtenir une estimation des distributions des temps de réponse d'une trame pour faire un premier dimensionnement "gros grains". En revanche, même si ces écarts sont faibles, et si l'analyse permet d'obtenir des probabilités pour des temps plus importants, cela rend l'utilisation peu adéquate pour valider des contraintes de sûreté. Dans [68], Zeng et al. font la même expérience pour la trame 25. Les courbes obtenues sont différentes de celles de la Figure 4.12 car leur approche ne prend pas en compte les offsets entre les trames émises entre les calculateurs et correspond donc à une configuration d'offsets entre les trames différente. De manière similaire à nos résultats, les courbes obtenues par analyse et simulation sont très proches mais présentent

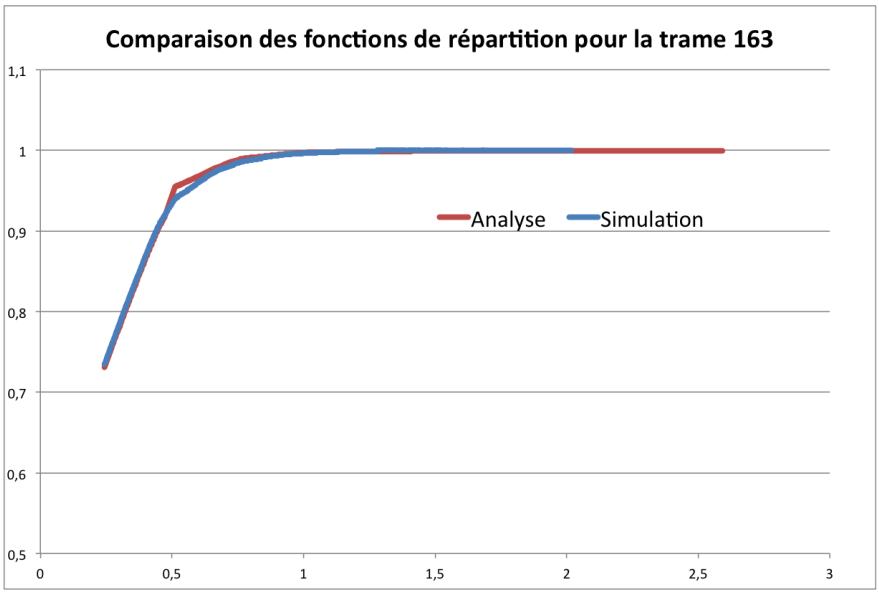

Figure 4.11 – Comparaison des fonctions de distributions pour la trame 163 (16ème dans l'ordre de priorité sur 68) du benchmark didactique de PSA obtenues par analyse (en rouge) et par simulation (en bleu). Dans cet exemple, la différence entre analyse et simulation est plus marquée. Cela s'explique par le fait que l'hypothèse selon laquelle il n'y a pas de trames plus prioritaire en attente au moment ou la trame 163 est prête à être émise n'est pas toujours respectée.

également de légers écarts. Contrairement à nos résultats, la courbe obtenue par analyse est en dessous de celle obtenue par simulation aux niveau des écarts. Cela signifie que leur approche sous-estime la probabilité des temps de réponses moyens alors que notre approche les sur-estime. En revanche, les deux courbes se rejoignent également sur la partie asymptotique dans les deux approches. Ceci indique que leur approche sur-estime la probabilité des grands temps de réponse et que notre approche les sous-estime.

Du point de vue des problèmes de vérification des contraintes de sûreté, notre approche présente néanmoins de l'intérêt car elle permet de formuler une borne inférieure sur la probabilité des grands temps de réponse. Pour cela, il faut être capable de déterminer à partir de quel moment les tendances s'inversent. Intuitivement, cela commence à se produire quand l'écart décroit entre les courbes, au fur et à mesure que les temps de réponse augmentent mais une démonstration formelle nécessitera de futures investigations.

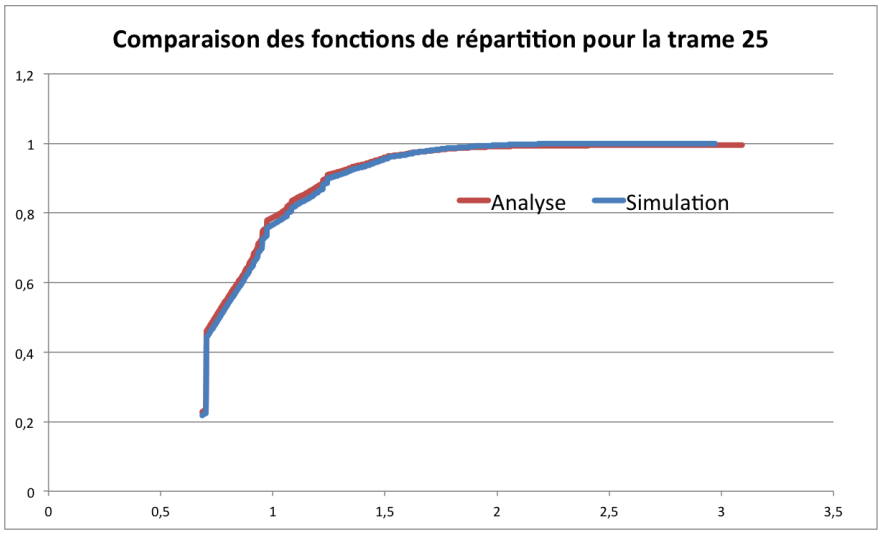

Figure 4.12 – Comparaison des fonctions de distributions pour la trame 25 (25 dans l'ordre de priorité sur 69) du benchmark issue de [68] obtenues par analyse (en rouge) et par simulation (en bleu). Les deux courbes sont très proches mais, comme prévu, nous constatons le même type d'écarts avant la partie asymptotique de la courbe.

4.4.2 Impact des paramètres d'approximation

La source principale de complexité dans l'algorithme est le nombre très important de scénarios d'interférences à interpréter. Ce nombre dépend directement du nombre de calculateurs émettant des trames de plus hautes priorités. Ainsi, le calcul de la taille de l'arbre, dont nous avons donné la formule en 4.3.5, indique que pour une messagerie composée de 10 calculateurs, le nombre de ces scénarios peut atteindre 24 millions pour les trames les moins prioritaires qui auront 9 calculateurs susceptibles d'interférer pendant un temps de blocage initial. Il faut ensuite interpréter chacun de ces scénarios pour toutes les valeurs possibles de blocage initial du bus par une trame de plus faible priorité : jusqu'à 135 fois si au moins une trame de 8 octets de donnée est présente dans les trames moins prioritaire et si la granularité utilisée et τ_{bit}. Enfin, chacun de ces 3 milliards de chemins résultera dans le calcul de nombreux temps de réponse résultant de la convolution des messages caractéristiques des calculateurs interférant. Par exemple, si chacun des messages caractéristiques des 9 calculateurs comporte au moins 3 valeurs de temps de transmission, cela peut faire jusqu'à $3^9 \approx 20000$ valeurs de temps de réponse pour chacun des chemins. À titre d'exemple pratique, le calcul de la distribution de temps de réponse de la trame 167 du benchmark didactique de PSA qui est susceptible d'être retardée par 8 autres calculateurs a nécessiter de calculer 55 milliards

de temps de réponse, ce qui a demandé 30h de calcul sur un processeur bi-coeur à 2,8 GHz.

Pour cette raison, nous avons introduit des paramètres d'approximations permettant de réduire le nombre de calculs à effectuer pendant l'analyse. Un premier moyen est d'utiliser une granularité moins fine pour les temps de blocage initiaux. Les deux autres moyens consistent à ne pas calculer les temps de réponse de probabilité trop faible et qui n'apporteront donc peu d'informations à la distribution de temps de réponse. Dans les expérimentations suivantes, nous étudions l'influence de ces paramètres sur la complexité des calculs et montrons qu'il est possible de réduire sensiblement le temps de calcul tout en obtenant des résultats identiques par rapport à une exploration exhaustive des scénarios. Pour cela, nous calculons par analyse la fonction de répartition de la trame 110 du benchmark didactique de PSA Peugeot Citroën sur un processeur i7 possédant 8 coeurs logiques cadencés à 2,0 GHz. Il s'agit de la $10^{ème}$ trame dans l'ordre de priorité pour laquelle 7 calculateurs émettent des trames plus prioritaires. Nous comparons ensuite pour chacun des cas le temps de calcul et la couverture de la distribution. Sans approximation, la fonction de répartition doit atteindre 100%. En pratique, la nature des calculs numériques peut faire perdre un peu de précision et cette valeur n'est jamais exactement atteinte.

Impact de la granularité

Dans cette première expérimentation, nous comparons le nombre de calcul à faire pour des granularités τ_{bit} et $5{\cdot}\tau_{bit}$ lors du calcul des temps de blocage initial du bus. Cela réduit effectivement le nombre de calculs à effectuer par 5. Les probabilités obtenues correspondent en conséquence à un intervalle de temps de réponse de longueur $5 \cdot \tau_{bit}$ au lieu des probabilités de chaque temps de réponse possible. De plus, la valeur $5 \cdot \tau_{bit}$ divise les valeurs de temps de blocage possible pour les trames de différentes longueurs de données. En conséquence, cela permet de capturer indépendamment chacun des sauts dans les fonctions de répartition et donc de conserver toutes les informations importantes contenues dans les distributions de temps de réponse.

Le tableau 4.1 donne les résultats obtenus pour les deux valeurs de granularité. Comme prévu, la granularité le plus fine a nécessité de calculer 5 fois plus de temps de réponse ce qui s'est traduit par un temps de calcul 3,5 fois plus grand que pour la plus grosse granularité. Le tracé de ces courbes montrerait qu'elles se superposent parfaitement à l'exception des quelques valeurs de temps de réponses les plus grandes car le temps de réponse maximum obtenu dans le second cas est inférieur de $4\tau_{bit}$. Le gain de performance est donc très important et ceci sans perdre d'information. En conséquence, nous utiliserons la granularité la plus grosse dans la suite des expérimentations.

Granularité	τ_{bit}	$5 \cdot \tau_{bit}$
Nbre de cas explorés (en millions)	3964	793
Temps de réponse max. calculé (en ms)	2,432	2,424
Couverture de la distribution (en %)	99,97%	99,97%
Temps de calcul	22m35s	6m32s

Table 4.1 – Influence du choix de la granularité pour l'utilisation initiale du bus sur le calcul de la distribution de temps de réponse de la trame 110 du benchmark didactique de PSA. Les calculs ont été réalisés sur un processeur intel i7 cadencé à 2.0 GHz.

Impact du seuil de probabilité

Dans cette seconde série d'expérimentations, nous étudions l'influence du seuil de probabilité sur le temps de calcul et la précision des résultats. Le tableau 4.2 donne une comparaison des résultats obtenus pour différentes valeurs de seuil de probabilité avec les résultats obtenus sans utiliser cette approximation. Une valeur de seuil de 10^{-12} permet déjà de diviser par 4 le nombre de scénarios étudiés et diviser par plus de 5 le temps de calcul sans perdre de précision de manière remarquable par rapport aux calculs réalisés sans utiliser le seuil d'approximation. À partir d'un seuil de valeur 10^{-10}, les résultats perdent 0,01% au niveau de la couverture de la distribution, ce qui est à peine remarquable. À partir d'un seuil de valeur 10^{-7}, la précision des résultats commence à baisser de manière plus sensible. Il est néanmoins intéressant de remarquer que seulement 21s de calcul auront été nécessaires pour obtenir 99,9% des temps de réponse alors que 6 minutes de calcul de plus (30 fois plus) sont nécessaires pour parcourir l'ensemble des possibilités. Enfin la valeur 10^{-11} semble être un excellent compromis pour cette série d'expérimentation car le temps de calcul est divisé par presque 7 fois sans perte de précision remarquable. Naturellement, le temps de réponse maximum calculé diminue avec la précision, mais ceci est tolérable dans la mesure où le pire temps de réponse théorique peut-être calculé par analyse et que les probabilités des temps de réponses entre le maximum calculé et la valeur pire cas sont infinitésimales.

Seuil de probabilité	0	10^{-12}	10^{-11}	10^{-10}	10^{-9}	10^{-8}	10^{-7}	10^{-6}	10^{-5}
Nbre de cas explorés (en millions)	793	220	**186**	151	120	92	69	51	36
Temps de réponse max. calculé (en ms)	2,424	2,024	**1,944**	1,754	1,674	1,484	1,214	1,154	0,944
Couverture de la distribution (en %)	99,97	99,97	**99,97**	99,96	99,95	99,91	99,80	99,50	98,64
Temps de calcul	6m32s	48s	**40s**	33s	27s	21s	17s	13s	11s

Table 4.2 – Influence du seuil de probabilité pour le calcul de la distribution de temps de réponse de la trame 110 du benchmark didactique de PSA. Les calculs ont été réalisés sur un processeur intel i7 cadencé à 2.0 GHz.

Impact du seuil de profondeur de l'arbre d'interférence

Dans cette nouvelle série d'expérimentations, nous étudions de manière similaire l'influence du seuil de profondeur pour le calcul des arbres d'interférence. La probabilité de n interférences consécutives de calculateur différents devient rapidement très faible. Eliminer ces scénarios peut permettre donc de réduire de manière efficace le temps de calcul. Ceci est illustré clairement dans les résultats présentés dans le tableau 4.3. Négliger les scénarios de plus de 4 étapes d'interférences permet de diviser par plus de 5 fois le temps de calcul sans perdre en précision. Nous avons expliqué précédemment qu'utiliser cette approximation induisait de l'erreur dans les résultats. Cependant lorsque les cas où l'erreur se produit sont rares au point de ne pas avoir d'influence sur la couverture de la distribution, l'erreur devient négligeable. En revanche, pour le cas où ce seuil est 2, non seulement il y a perte d'informations mais en plus une partie des informations calculées sont victimes d'un biais comme expliqué en section 4.3.7.

Seuil de profondeur de l'arbre d'interférence	max=7	6	5	**4**	3	2
Nbre de cas explorés (en millions)	793	720	482	**195**	39	3
Couverture de la distribution (en %)	99,97	99,97	99,97	**99,97**	99,94	99,59
Temps de calcul	6m32s	3m58s	2m25s	**50s**	9s	1s

Table 4.3 – Influence de la limite de précision pour le calcul de la distribution de temps de réponse de la trame 110 du benchmark didactique de PSA. Les calculs ont été réalisés sur un processeur intel i7 cadencé à 2.0 GHz.

Impact combiné des seuils de précision et de profondeur

Dans cette dernière série d'expérimentations, nous combinons tous les paramètres d'approximation en utilisant les valeurs autour des meilleurs compromis trouvés pour chacun de ces paramètres dans les expérimentations précédentes. Les résultats sont données dans le tableau 4.4.

Limite de précision	0	10^{-12}	10^{-12}	10^{-12}	10^{-11}	10^{-11}
Limite de profondeur de l'arbre d'interférence	max	6	5	**4**	5	4
Nbre de cas explorés (en millions)	793	213	176	**98**	149	83
Temps de réponse max. calculé (en ms)	2,424	2,024	2,024	**2,024**	1,944	1,944
Couverture de la distribution (en %)	99,97	99,97	99,97	**99,97**	99,96	99,96
Temps de calcul	6m32s	46s	36s	**18s**	29s	15s

Table 4.4 – Influence de la limite de précision pour le calcul de la distribution de temps de réponse de la trame 110 du benchmark didactique de PSA. Les calculs ont été réalisés sur un processeur intel i7 cadencé à 2.0 GHz.

À la lumière des résultats obtenus, il paraît donc intéressant d'utiliser simultanément les différents paramètres d'approximation. Utiliser un seuil de probabilité égal à 10^{-12} et un seuil de profondeur égal à 4 pour la profondeur de l'arbre d'interférence a permis de réduire le temps de calcul à 18s sans perdre de précision de manière remarquable. Pour une granularité égale à $5\tau_{bit}$, cela a permis de calculer 8 fois moins de cas et d'être 15 fois plus rapide par rapport à un calcul n'utilisant pas ces paramètres. Pour une granularité égale à τ_{bit}, cela permet d'explorer 40 fois moins de cas et d'être 75 fois plus rapide, ce qui est considérable. La Figure 4.13 permet de comparer les courbes obtenues avec et sans l'utilisation de ces paramètres. À l'exception des temps de réponses les plus importants, les deux courbes se superposent parfaitement. La perte des valeurs de probabilités des temps de réponse les plus importants est généralement acceptable dans la mesure où leurs probabilités sont négligeables et que le temps de réponse pire cas peut-être obtenu rapidement par d'autres types d'analyse.

La valeur optimale des paramètres peut néanmoins varier légèrement d'une trame à une autre. Utiliser ces valeurs pour la trame 167 a résulté en une perte de 0,4% de la couverture de la distribution de temps de réponse. En revanche, ceci a permis de finir les calculs en 6 minutes et 40s sur le processeur i7 à 2.0 GHz, à comparer aux 30h nécessaires sans utiliser les paramètres d'approximation sur un core2 duo à 2,8 GHz (qui était cependant beaucoup moins performant car il

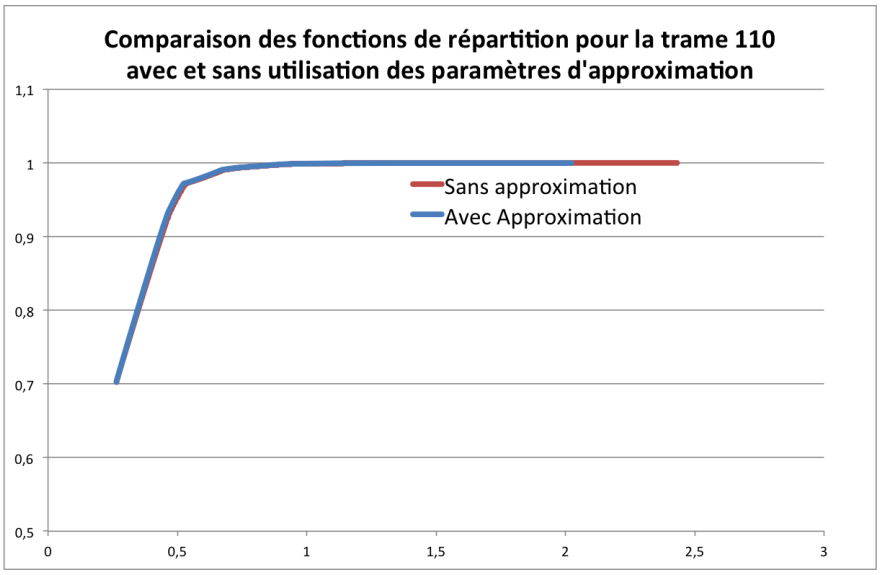

Figure 4.13 – Comparaison des courbes obtenues en effectuant les calculs sans utiliser les paramètres d'approximation (en rouge) et en utilisant les paramètres d'approximation (en bleu). La courbe bleue est obtenue en utilisant une granularité égale à $5\tau_{bit}$, un seuil de probabilité égal à 10^{-12} et un seuil de profondeur de l'arbre d'interférence égal à 4. Utiliser ces approximations a permis d'obtenir la fonction de répartition de temps de réponse de la trame étudié en 75 fois moins de temps.

possédait moins de coeurs de calculs).

4.4.3 Avantages et inconvénients de l'analyse et de la simulation

Les précédentes expérimentations ont permis de mettre en évidence les avantages et inconvénients respectifs de l'analyse et de la simulation. La volonté de mettre au point une approche analytique était motivée par le problème d'exhaustivité d'une approche par simulation. De manière attendue, l'analyse développée dans ce chapitre permet de calculer plus facilement des valeurs de probabilités pour les temps de réponses trop rares pour être observés régulièrement par simulation. En revanche, l'analyse a deux inconvénients par rapport à la simulation. Tout d'abord, elle souffre d'un problème de passage à l'échelle. Le nombre de calculs dépend très fortement du nombre de calculateurs communicants sur le réseau et le nombre de scénarios à considérer explose très rapidement. Une utilisation judicieuse des paramètres d'approximation permet, dans une certaine mesure, de réduire le problème mais nous n'avons pas été capable d'effectuer une analyse complète pour plus de 10

calculateurs communicant sur un réseau. En revanche, il est très facile d'obtenir ces distributions de temps de réponses par simulation, pour un nombre beaucoup plus importants de calculateurs. Pour les messageries avec peu de calculateurs, l'analyse peut-être plus rapide et calculer plus de valeurs que la simulation mais il devient assez vite intéressant d'effectuer des simulations, d'autant plus qu'une simulation permet d'obtenir simultanément les distributions de temps de réponse de toutes les trame. Le deuxième inconvénient de l'analyse est lié à l'utilisation de l'hypothèse stipulant qu'aucune trame plus prioritaire est en attente d'être émise au moment où a trame étudiée commence à participer au processus d'arbitration. Si le contraire est effectivement rare pour les trames les plus prioritaires, cette hypothèse se révèle peu pratique et introduit un biais dans les probabilités calculées, en particulier si la charge du réseau est importante. Cette erreur est source d'optimisme pour les temps de réponse les plus faibles mais de pessimismes pour la queue de distribution. Néanmoins, ce dernier aspect peut-être intéressant pour vérifier des contraintes de sûreté mais n'a pas été formellement démontré.

4.5 Conclusion

Dans ce chapitre nous avons présenté une approche analytique permettant d'évaluer les distributions de temps de réponse des trames CAN. Cette approche vient compléter l'approche de simulation présentée dans le chapitre 3 qui ne permet pas de manière fiable de reproduire les temps de réponses les plus importants. Le calcul exhaustif de la distribution de temps de réponses transmises sur un bus avec N calculateurs demanderait de considérer $(\frac{PPCM(C_i)}{\tau_{bit}})^N \approx (5.10^5)^N$ scénarios de déphasages entre calculateurs si le PPCM des périodes des trames est $1s$ et pour un débit de 500 kbit/s. Pour parer à ceci, l'utilisation du concept de message caractéristique emprunté à [68] et la formulation de certaines hypothèses permettent de réduire le nombre de calculs et d'obtenir plus facilement une estimation de la distribution des temps de réponse. Nous introduisons de plus des paramètres d'approximation permettant de réduire de manière remarquable les temps de calcul pour un sacrifice négligeable de précision s'ils sont choisi judicieusement. Nous avons obtenu expérimentalement des résultats par analyse très proches de ceux obtenus par simulation. Ces expérimentations ont permis de mettre en évidence les avantages et inconvénients respectifs des deux approches. Dans les cas favorables, avec un nombre réduit de calculateurs, l'analyse peut permettre d'obtenir des résultats plus riches et plus rapidement que par simulation. Cependant, l'approche par analyse souffre d'un problème de passage à l'échelle absent de l'approche par simulation. De plus, l'hypothèse selon laquelle aucune trame plus prioritaire est en attente d'être émise au moment où la trame étudiée commence à participer au processus d'arbitration est parfois un peu forte pour les bus chargés et les trames de faible

priorité ce qui pour conséquence d'induire un biais dans les résultats.

L'étude de cette hypothèse ouvre un certain nombre de perspectives pour de futurs travaux. En effet, il serait intéressant de voir comment prendre en compte de manière plus fine le backlog initial (les trames plus prioritaires en attente d'être émise). Cependant, cela nécessiterait plus de calculs alors que l'analyse présentée est déjà très complexe. Enfin, l'erreur induite par cette hypothèse semble être source de sur-estimation pour les temps de réponses faibles, pour lesquels notre approche calcule une probabilité plus forte que dans la réalité, mais est source de sous-estimation pour les temps de réponses les plus importants (qui serait en réalité plus fréquents). Cette sous-estimation de probabilité des temps de réponses les plus importants permet de donner une borne inférieure sur la probabilité de ces événements redoutés et peut donc être utile lors de la validation de contraintes de sûreté exprimées de manière probabiliste. Ceci nécessitera cependant des travaux supplémentaires afin d'identifier formellement le temps de réponse à partir duquel ce phénomène se produit.

Chapitre 5

Conclusion

Vérification des contraintes de temps de bout-en-bout dans le contexte AUTOSAR

Aujourd'hui les systèmes électroniques embarqués dans les véhicules ont une complexité sans cesse croissante. Lors de la conception d'un système distribué pour l'automobile, il est crucial d'en maîtriser le comportement dynamique et les différents paramètres temporels afin de garantir la sécurité ainsi que le confort des passagers du véhicule. La vérification des contraintes temporelles est donc un enjeu majeur dès les première phases de développement des fonctions distribuées sur les différents calculateurs au sein de l'architecture électronique embarquée dans les voitures. La maîtrise du comportement temporel des fonctions pilotées par électronique contribue de plus à résoudre les autres grands challenges dans ce domaine que ce sont l'inter-opérabilité, la sûreté fonctionnelle et l'optimisation. Enfin, l'industrie automobile a adopté très largement le standard AUTOSAR pour l'architecture logicielle des calculateurs embarqués au point d'en devenir incontournable. Les travaux de cette thèse contribuent donc à résoudre le problème de la vérification des contraintes de temps de bout-en-bout dans le contexte AUTOSAR.

Dans la pratique, exhiber une méthode de vérification des contraintes de bout-en-bout est difficile ou impossible dans le contexte AUTOSAR, en particulier si on ajoute les contraintes liées au processus de développement automobile. En effet, les différentes étapes du développement sont partagées entre plusieurs acteurs et réalisées en parallèle. Ainsi la conception de l'architecture matérielle, le développement des fonctions applicatives et la spécification des messageries des réseaux de communication sont couramment effectués en parallèle. Ceci est permis par l'architecture logicielle de référence AUTOSAR qui repose sur un asynchronisme entre les activités implémentant les fonctions applicatives et les activités en charge de la communication entre les calculateurs. Cela se traduit aussi par la présence de composants boîtes noires achetés à des fournisseurs. En conséquence, il est difficile

d'avoir toutes les informations nécessaires pour couvrir une contrainte temporelle de bout-en-bout. Le manque de ces informations et les dépendances complexes entre variables partagées rendent impossible une approche de type holistique où l'on chercherait à effectuer une analyse prenant en compte l'ensemble de la chaîne de sous-contraintes étudiées.

Au fur et à mesure du processus de développement, les contraintes temporelles de bout-en-bout, exprimées au départ sous la forme d'une borne sur un temps de réponse entre un événement au niveau d'un capteur et une réaction d'un actionneur, sont alors décomposées en une séquence de contraintes intermédiaires. Ces sous-contraintes sont appliquées à l'exécution des fonctions mises en jeu ainsi qu'aux délais de transmission des données entre celles-ci. Comme il est irréaliste en pratique de traiter le problème de bout-en-bout, nous étudions donc ces différentes contraintes intermédiaires. Pour cette raison, les travaux de thèse s'intéressent plus particulièrement à étudier indépendamment les délais au niveau des composants logiciels applicatifs d'une part et de la transmission de message sur les bus de communication CAN d'autre part. À l'exception des délais au niveau de l'interface entre ces deux niveaux, la synthèse des informations obtenues ainsi permet alors de reconstituer le chemin suivie par le signal correspondant à une contrainte de bout-en-bout.

Contributions

Les travaux de thèse contribuent au développement de systèmes sûrs embarqués dans l'automobile vérifiant leurs contraintes temporelles et permettent en particulier :

- minimiser le coût des systèmes embarqués via le lissage de la charge périodique sur les calculateurs permettant un "downsizing" des processeurs
- maximiser la charge de l'ensemble des fonctions embarquées dans la mesure où le lissage de la charge périodique permet d'ajouter de manière efficace de nouvelles fonctions sur une architecture existante sans nécessairement modifier l'architecture matérielle
- une connaissance plus fine du comportement temporel des trames CAN via l'obtention de leurs distributions de temps de réponses par simulation ou analyse plutôt que leurs pires cas de temps de réponse.

Dans un premier temps, nous avons présenté une approche permettant d'améliorer l'utilisation des calculateurs exécutant un grand nombre de composants logiciels compatible avec l'introduction progressive des plateformes multi-coeur. Nous avons décrit des algorithmes rapides et efficaces pour lisser la charge périodique sur les calculateurs multi-coeurs en adaptant puis en améliorant une approche existant pour les bus CAN qui déphase les messages en calculant des offsets. Nous avons également donné des résultats théoriques sur l'efficacité des algorithmes dans cer-

tains cas particuliers. Enfin, nous avons discuté les possibilités d'utilisation de ces algorithmes en fonction des autres tâches exécutées sur le calculateur.

La suite des travaux a été consacrée à l'étude de distributions de temps de réponse pour les messages transmis sur les bus CAN entre les calculateurs. Dans un premier temps nous nous sommes consacrés à la mise au point d'une approche de simulation basée sur les dérives d'horloges des calculateurs transmettant sur le réseau. Ce phénomène est inévitable et généralement négligé lors de l'étude des bus CAN. Cette approche de simulation a ensuite été utilisée pour étudier ce qu'il se passe autour de l'occurrence des pires cas de temps de réponse et enfin pour étudier les distributions de temps de réponses obtenues pour de grandes durées de temps simulé. Nous avons ainsi montré, en reproduisant les scénarios provoquant des temps de réponse pire cas pour des trames de faibles priorité, que les situations causant d'importants temps de réponse s'estompent assez rapidement (après quelques hyper-périodes du système étudié). Pour de longues durées de simulation, nous avons montré que les déphasages initiaux entre les calculateurs n'ont pas d'influence sur les distributions de temps de réponses obtenues, à l'exception des valeurs maximales constatées. Ensuite nous avons montré que les valeurs des dérives tolérées dans l'industrie influent de manière négligeable sur les distributions de temps de réponses. Enfin, nous avons montré que nous obtenons des distributions de temps de réponse identiques en réalisant une longue simulation avec des dérives d'horloge ou en faisant un grand nombre de courtes simulations sans dérives d'horloge en changeant les déphasages initiaux entre les calculateurs, pratique répandue pour obtenir une distribution de temps de réponse d'une trame CAN.

L'obtention par simulation d'une distribution de temps de réponse unique, avec et sans dérive d'horloge est un résultat important car cela justifie l'utilisation de tels objets pour décrire le comportement temporel des trames CAN de manière beaucoup plus riche qu'avec un temps de réponse pire cas. L'inconvénient des simulations est leur non-exhaustivité. Nous avons donc étudié dans le dernier chapitre une technique analytique pour évaluer les distributions de temps de réponse. Sur la base d'observations de résultats obtenus par simulation, nous proposons une approche de calcul de ces distributions. Nous présentons également différents paramètres d'approximation permettant de réduire le nombre très important de calcul à effectuer en limitant la perte de précision. Enfin, nous comparons expérimentalement les résultats obtenus par analyse et simulation afin d'observer que l'analyse présentée donne des résultats proches de la simulation même si des différences apparaissent pour des valeurs élevées de charge du bus.

En rapport avec les travaux de recherches présentés dans les chapitres 2, 3 et 4, les annexes B et C contiennent respectivement l'évaluation d'un outil logiciel de calcul des pires cas de temps d'exécution (nécessaires au lissage de charge) et la description de dispositifs brevetés [7, 8, 9] pour résoudre les situations d'encombrement des réseaux de communication (en particulier de type CAN).

Contribution des résultats à l'industrie automobile

Dans le contexte du processus de développement d'un constructeur comme PSA Peugeot Citroën, les résultats de la thèse apportent des éléments de décision afin de vérifier et optimiser l'architecture électronique embarquée dans les véhicules. En particulier, les deux derniers chapitres apportent des contributions importantes dans la perspective d'allonger la durée de vie de ces architectures. Tout d'abord, les garanties de sûreté de fonctionnement sont à présent envisageables à partir de garanties probabilistes. De plus, en ce qui concerne spécifiquement CAN, les travaux de thèses et les brevets associés contribuent à pouvoir utiliser plus longtemps cette technologie avant de devoir faire la transition vers d'autres technologie pour les réseaux de communication comme FlexRay ou Ethernet switché.

Les résultats des travaux de thèse sont aussi étroitement liés au projet TIMMO-2-USE [2] étudiant les aspects temporels dans le processus de conception de l'industrie automobile et qui s'est achevé en septembre 2012. Suite à la première itération du projet TIMMO, des contributions significatives avaient déjà été faites dans le domaine de méthodologie et de la prise en compte des propriétés temporelles dans les langages de description d'architecture électronique (correspondant respectivement aux groupes de travail WP2 et WP4 de TIMMO-2-USE). Les résultats de thèse s'inscrivent plutôt dans le contexte du groupe de travail WP3 du projet sur les outils et les algorithmes et servent de support à certains des cas d'usages principaux définis au début du projet et détaillés dans les livrables du projet (en libre accès sur le site internet du projet [2]) . En particulier, l'optimisation de l'utilisation des calculateurs et l'expression de garanties temporelles probabilistes contribuent au cas d'utilisation "Négocier les budgets temporels" (UC#4). L'obtention des distributions de temps de réponses des trames CAN contribuent également au cas d'utilisation "Spécifier des propriétés temporelles probabilistes" (UC#11). Enfin, l'ensemble des travaux de thèses contribuent également fortement au cas d'utilisation "spécifier le dimensionnement du système" (UC#13).

Perspectives

Dans l'objectif de pouvoir faire la synthèse de bout-en-bout il manque encore des solutions pour maîtriser et vérifier les délais correspondant aux asynchronismes introduits par AUTOSAR OS entre le couche de communication et la couche des "Application Software Components". En effet, il serait intéressant de travailler sur une alternative à la pratique actuelle consistant à considérer les pires cas d'asynchronismes en comptant une période de l'activité consommant l'information produite (correspondant au cas où l'information consommée est disponible immédiatement après le début d'une instance de l'activé utilisant celle-ci).

Une autre limite du travail, imposée par le processus de développement, est liée aux variables partagées utilisées par les fonctions intermédiaires. Prendre en

compte de manière exhaustive ces variables et leurs dépendances permettrait de raffiner les pratiques d'analyse temporelle dans l'industrie automobile. Une première conséquence d'une telle approche serait que la décomposition d'une contrainte de temps entre le capteur et l'actionneur ne serait plus linéaire. Cela soulèverait ainsi des problèmes de synchronisation entre plusieurs signaux et de fraîcheur de données. Il faudrait donc élargir la définition de contrainte temporelle, restreinte ici à des bornes sur des temps de réponse, à d'autres cas pouvant éventuellement mettre en cause plus qu'un seul signal.

Appendix A

Messageries CAN utilisées

A.1 SAE Benchmark

ID	Description du signal	Données (octets)	Période (ms)	Echéance (ms)	Emetteur
1	Traction Battery Voltage	8	100	100	Battery
2	Traction Battery Current	8	100	100	Battery
3	Traction Battery Temp, Average	8	1000	1000	Battery
4	Auxiliary Battery Voltage	8	100	100	Battery
5	Traction Battery Temp, Max	8	1000	1000	Battery
6	Auxiliary Battery Current	8	100	100	Battery
7	Accelerator Position	8	5	5	Driver
8	Brake Pressure, Master Cylinder	8	5	5	Brakes
9	Brake Pressure, Line	8	5	5	Brakes
10	Transaxle Lubrication Pressure	8	100	100	Trans
11	Transaction Clutch Line Pressure	8	5	5	Trans
12	Vehicle Speed	8	100	100	Brakes
13	Traction Battery Ground Fault	1	1000	1000	Battery
14	Hi&Lo Contactor Open/Close	4	50	5	Battery
15	Key Switch Run	1	50	20	Driver
16	Key Switch Start	1	50	20	Driver
17	Accelerator Switch	2	50	20	Driver
18	Brake Switch	1	20	20	Brakes
19	Emergency Brake	1	50	20	Driver
20	Shift Lever (PRNDL)	3	50	20	Driver
21	Motor/Trans Over Temperature	2	1000	1000	Trans
22	Speed Control	3	50	20	Battery

Table A.1: 22 premiers messages de la messagerie isssue du SAE Benchmark [4]. Le débit du réseau est 512 Kbit/s

ID	Description du signal	Données (octets)	Période (ms)	Echéance (ms)	Emetteur
23	12V Power Ack Vehicle Control	1	50	20	Battery
24	12V Power Ack Inverter	1	50	20	Battery
25	12V Power Ack I/M Contr	1	50	20	Battery
26	Brake Mode (Parallel/Split)	1	50	20	Driver
27	SOC Reset	1	50	20	Driver
28	Interlock	1	50	20	Battery
29	High Contactor Control	8	10	10	V/C
30	Low Contactor Control	8	10	10	V/C
31	Reverse and 2nd Gear Clutches	2	50	20	V/C
32	Clutch Pressure Control	8	5	5	V/C
33	DC/DC Converter	1	1000	1000	V/C
34	DC/DC Converter Current Control	8	50	20	V/C
35	12V Power Relay	1	50	20	V/C
36	Traction Battery Ground Fault Test	2	1000	1000	V/C
37	Brake Solenoid	1	50	20	V/C
38	Backup Alarm	1	50	20	V/C
39	Warning Lights	7	50	20	V/C
40	Key Switch	1	50	20	V/C
41	Main Contactor Close	1	50	20	I/M C
42	Torque Command	8	5	5	V/C
43	Torque Measured	8	5	5	I/M C
44	FWD/REV	1	50	20	V/C
45	FWD/REV Ack	1	50	20	I/M C
46	Idle	1	50	20	V/C
47	Inhibit	1	50	20	I/M C
48	Shift in Progress	1	50	20	V/C
49	Processed Motor Speed	8	5	5	I/M C
50	Inverter Temperature Status	2	50	20	I/M C
51	Shutdown	1	50	20	I/M C
52	Status/Malfunction (TBD)	8	50	20	I/M C
53	Main Contactor Acknowledge	1	50	20	V/C

Table A.2: Fin de la messagerie issue du SAE Benchmark [4].

A.2 Benchmark from Zeng et al.[68]

ID	Données (octets)	Période (ms)	Emetteur	ID	Données (octets)	Période (ms)	Emetteur
1	8	10	ECU_2	36	6	25	ECU_6
2	8	10	ECU_2	37	8	50	ECU_4
3	4	5	ECU_3	38	8	50	ECU_4
4	7	10	ECU_3	39	8	50	ECU_5
5	4	10	ECU_1	40	5	50	ECU_3
6	8	10	ECU_6	41	8	50	ECU_4
7	8	100	ECU_1	42	8	50	ECU_6
8	2	100	ECU_1	43	8	50	ECU_2
9	3	100	ECU_1	44	8	100	ECU_5
10	8	25	ECU_5	45	2	25	ECU_2
11	2	100	ECU_1	46	4	50	ECU_2
12	4	20	ECU_1	47	4	50	ECU_2
13	4	100	ECU_2	48	8	100	ECU_4
14	4	100	ECU_1	49	8	100	ECU_3
15	8	100	ECU_5	50	8	100	ECU_1
16	6	10	ECU_1	51	1	100	ECU_3
17	7	100	ECU_2	52	8	100	ECU_3
18	8	100	ECU_2	53	4	100	ECU_1
19	7	50	ECU_2	54	1	100	ECU_3
20	8	10	ECU_3	55	1	100	ECU_3
21	8	10	ECU_6	56	8	100	ECU_5
22	8	25	ECU_6	57	7	100	ECU_3
23	6	25	ECU_3	58	1	100	ECU_3
24	6	25	ECU_3	59	1	100	ECU_3
25	7	25	ECU_3	60	3	100	ECU_4
26	8	20	ECU_2	61	8	100	ECU_1
27	8	25	ECU_5	62	1	100	ECU_1
28	8	20	ECU_2	63	4	100	ECU_3
29	3	25	ECU_1	64	8	100	ECU_1
30	5	10	ECU_2	65	1	100	ECU_1
31	8	20	ECU_2	66	1	100	ECU_1
32	8	10	ECU_4	67	8	50	ECU_2
33	8	10	ECU_3	68	1	100	ECU_1
34	8	10	ECU_3	69	8	100	ECU_4
35	7	25	ECU_1				

Table A.3: Un exemple de messagerie CAN avec 6 ECUs et 69 messages provenant de [68]

Annexe B

Évaluation de l'outil aiT d'AbsInt pour le calcul de WCET

Sommaire

B.1 Référence, glossaire

B.1.1 Références

– Rapport final du WCET Tool Challenge 2006, Lily Tan « 2nd International Symposium on Leveraging Applications of Formal Methods » (ISoLA'06), Cyprus
– "AbsInt Advanced Analyzer for PowerPC MPC55xx User Documentation", AbsInt
– "Guidelines PowerPC MPC55xx Hardware Configuration to Improve Precision of aiT WCET Analysis Results", AbsInt
– "e200z6 PowerPC™ Core Reference Manual", Freescale Semiconductors
– "MPC5553/5554 Microcontroller Reference Manual", Freescale Semiconductors
– "MPC5554 Microcontroller Data Sheet", Freescale Semiconductors

B.1.2 Glossaire

WCET : Worst Case Execution Time, c'est le pire temps d'exécution

ECU : Electronic Control Unit

B.2 Contexte de l'étude

B.2.1 Objet de l'étude

L'objet de cette étude est d'évaluer l'outil de calcul de pire temps d'exécution « aiT » développé par la société allemande AbsInt. Cet outil utilise des méthodes d'analyse statique du code machine compilé et lié afin de calculer une borne supérieure sûre du pire temps d'exécution du code étudié. Le but est d'évaluer les

performances et les difficultés d'utilisation de ces méthodes et de cet outil pour du code embarqué dans les véhicules automobiles.

Dans le cadre des travaux de thèse, nous nous sommes intéressés à l'évaluation du pire temps d'exécution de code embarqué sur des ECU. Au moment de cette étude, l'outil aiT d'AbsInt semblait un des outils commerciaux les plus prometteurs. En effet, cet outil s'était illustré par rapport à ses concurrents lors du WCET tool challenge 2006. De plus, aiT a été utilisé pour des cas industriels en avioniques, en particulier lors du développement de l'A380 chez Airbus. Notons toutefois qu'au meilleur de notre connaissance, l'outil ne semble être utilisé que de façon exceptionnelle dans le cadre du domaine électronique dans le secteur de l'automobile.

B.2.2 Les pires temps d'exécution

Le pire temps d'exécution, ou WCET, d'un fragment de code sur un microprocesseur donné correspond au temps maximum nécessaire au microprocesseur pour exécuter ce fragment de code, quelque soit les paramètres possibles utilisés en entrée du fragment de code ou l'état du microprocesseur (état du cache et du pipeline en particulier). Pour les pires temps d'exécution, il est fait l'hypothèse de l'absence d'interruption pendant l'exécution du code étudié.

En pratique, ce pire temps d'exécution peut être très supérieur au temps d'exécution moyen pour un même fragment de code sur un même microprocesseur. Cependant, la connaissance du pire temps d'exécution est nécessaire aux méthodes d'analyse pire cas des propriétés temporelles des fonctions véhicules pilotées ainsi que pour la plupart des méthodes d'ordonnancement de tâches sur les ECU. La connaissance de ce pire temps d'exécution peut servir à dimensionner l'architecture électronique du véhicule ou valider le choix d'un microprocesseur. De plus, la connaissance d'une borne supérieure du pire temps d'exécution est indispensable à toute approche de certification de l'architecture électronique (en particulier en perspective de la norme ISO 2626-2).

B.2.3 Présentation de l'outil

L'outil de calcul des pire temps d'exécution est distribué au sein d'une interface logicielle se nommant a3 (AbsInt Advanced Analyzer). Cette dernière réunit plusieurs outils dont aiT pour l'analyse des pires temps d'exécution, Stack pour l'analyse de l'utilisation de la pile et aiSee qui permet d'afficher des graphes. Dans le cadre de cette étude, nous nous sommes concentrés sur aiT.grandeurs. Contrairement aux méthodes basées sur des mesures et des simulations, l'analyse statique est exhaustive par nature et non basée sur un ensemble discret de scénario. Cependant, les connaissances solides du code étudié et de l'outil aiT sont nécessaires à l'obtention de résultats satisfaisants.

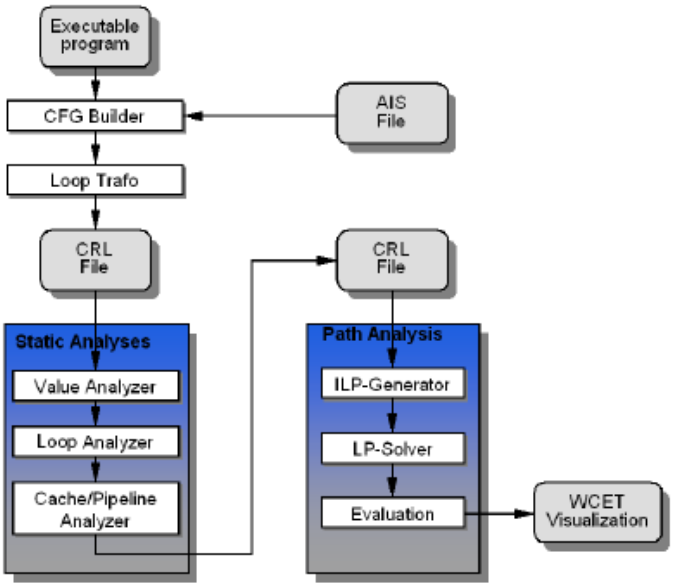

Figure B.1 – Structure du logiciel aiT.

aiT utilise des techniques d'analyse statique reposant sur l'interprétation abstraite pour calculer les pire temps d'exécution. Ceci a permis d'améliorer considérablement l'analyse du cache et du pipeline traditionnellement sujette à des surestimations de plusieurs ordres de grandeurs. Contrairement aux méthodes basées sur des mesures et des simulations, l'analyse statique est exhaustive par nature et non basée sur un ensemble discret de scénario. Cependant, les connaissances solides du code étudié et de l'outil aiT sont nécessaires à l'obtention de résultats satisfaisants.

B.2.4 Structure de aiT

Le schéma de la Figure B.1 illustre les différentes étapes de calcul réalisées par l'outil d'AbsInt.

Ces différentes étapes sont à présent brièvement décrites. Pour des explications précises sur le contenu et les techniques utilisées, le lecteur est invité à se référer à la documentation du logiciel, les supports de formations et les articles scientifiques se rapportant aux différentes techniques.

En entrée, il faut fournir à l'outil le code assemblé et lié à analyser ainsi qu'un fichier d'annotations permettant d'aider l'outil à réaliser ou affiner le calcul du pire temps d'exécution. De plus, il est nécessaire de fournir différents paramètres décrivant la configuration de la cible (fréquence du processeur et structure du cache entre autres. . .).

En premier temps, le CFG Builder reconstruit le graphe de flux de contrôle du programme ("Control Flow Graph"), c'est-à-dire l'enchaînement des instructions atteignables depuis l'entrée du programme en tenant compte des différents branchements possibles. L'outil procède ensuite à une transformation des boucles et des fonctions récursives : il "déroule" explicitement les premières itérations suivant une profondeur paramétrable. Ce déroulement est nécessaire pour distinguer les différentes itérations d'une boucle/fonction récursive lors de l'analyse. Néanmoins, le déroulement peut être partiel afin de simplifier les calculs dans les cas où les itérations sont identiques à partir d'un certain seuil. Ici, l'utilisateur peut choisir de privilégier la précision ou la vitesse d'analyse en jouant sur les paramètres de déroulement des boucles.

Les trois étapes suivantes reposent sur des méthodes d'interprétation abstraite. En particulier, l'outil utilise le principe d'intervalles abstraits (« **Abs**tract **Int**ervals ») et interprète dans quels intervalles varient les différentes valeurs utilisées pour les calculs. Cette méthode permet d'obtenir des résultats dans un temps raisonnable pour des calculs infaisables s'il fallait essayer toutes les valeurs une par une (en particulier quand les intervalles ne sont pas bornés). Néanmoins, le fait d'utiliser cette technique d'abstraction implique nécessairement des pertes d'informations et donc une diminution de la précision. Il s'agit d'une analyse pire cas, l'hypothèse la plus pessimiste est donc toujours choisie.

Les étapes d'analyse statique consistent successivement à déterminer le nombre d'itérations des boucles ou fonctions récursives, puis à déterminer les adresses pour tous les accès mémoire et ensuite à analyser le comportement du pipeline et du cache de la cible étudiée lors de l'exécution des différentes séquences d'instructions.

Enfin, la dernière étape consiste à trouver le chemin correct le plus long dans le graphe de flux de contrôle du programme. Il s'agit d'un problème d'optimisation sous contraintes. Pour le résoudre, l'outil utilise un solveur de problème de programmation linéaire en nombres entiers.

Les annotations données par l'utilisateur dans le fichier AIS permettent d'ajouter des contraintes pour toutes les différentes analyses afin de permettre de trouver un résultat et/ou de l'affiner.

B.3 Jeux d'essai

B.3.1 La cible

Dans le cadre de cette évaluation, les mesures et calculs ont été effectués pour le processeur MPC5554 de Freescale. Il s'agit d'un Power PC basé sur le cœur e200z6 et muni de 64 Kb de RAM et de 2 Mb de Flash. Le cœur e200z6 était configuré pour opérer à 128 MHz. Suivant les essais, le cache de 32 Kb était activé ou désactivé.

B.3.2 Le code

Pendant cette étude, six fonctions codées en C ont été étudiées avec aiT. Les trois premières sont extraites du benchmark de l'université de Malärdalen pour le calcul de pire temps d'exécution et les trois dernières proviennent de PSA. Deux d'entre elles ont été anonymisées car elles sont utilisées en production. L'objectif principal était d'essayer l'outil sur du code provenant de PSA mais les fonctions de Malärdalen sont aussi utiles pour valider le protocole utilisé pour l'étude.

Le code C des fonctions développées en internes a été généré automatiquement à partir de modèles Simulink au moyen de l'outil TargetLink.

B.3.3 Cnt

La fonction "Cnt" est issue du benchmark de Malärdalen. Cette fonction compte les nombres non-négatifs dans une matrice de taille fixe dont les nombres sont générés aléatoirement. Cette fonction contient des boucles imbriquées. Le nombre d'itération est fixe d'une exécution sur l'autre car il ne dépend que de la taille de la matrice.

B.3.4 Ndes

La fonction "Ndes" est aussi extraite du benchmark de Malärdalen. Il s'agit de code complexe écrit pour une application embarquée. Cette fonction comprend des opérations sur les bits (e.g. des décalages), et sur des tableaux ainsi que des calculs sur des matrices. Le programme comprend 12 boucles non imbriquées. Le nombre d'itérations des boucles dépend des valeurs d'entrée.

B.3.5 Statemate

Enfin, le dernier programme provenant de ce benchmark, "Statemate", est issu de génération automatique de code à partir d'un modèle Statemate. La génération de code a été réalisée à l'aide de l'outil STARC (STAtechart Real-time-Code generator). Le nombre d'itération des boucles du programme dépend de l'entrée du programme pour lequel 80 variables externes sont impliquées.

B.3.6 PSA#1

Cette première fonction développée chez PSA ne comporte pas de boucles ni d'appels à d'autres fonctions. Elle comporte un nombre très important de branchements conditionnels et un switch. Le flux de contrôle de cette fonction est donc particulièrement complexe.

B.3.7 PSA#2

Cette seconde fonction présente aussi un nombre important de branchements conditionnels. De plus, elle fait appel à des fonctions correspondant à des cartographies et qui contiennent des boucles. Enfin, cette fonction fait également appel à des librairies externes de TargetLink pour certains calculs.

B.3.8 Ref-model

Cette dernière fonction provient d'un modèle didactique. Elle fait également appel à des cartographies mais sa structure est plus simple que celle des deux autres programmes développés en interne.

B.4 Protocole de test

B.4.1 Préparation du code

Afin de pouvoir l'exécuter sur une cible réelle, il était nécessaire d'ajouter dans le code des programmes-test les instructions nécessaires à la configuration du MPC5554. Ainsi, le code final contenait une première série d'instructions permettant de paramétrer la cible puis une seconde série d'instructions qui appelait en boucle un des programmes du jeu d'essai.

La première série d'instructions permettait de changer les valeurs par défaut de différents registres configuration de la carte d'évaluation. Tout d'abord, la fréquence du cœur était réglée à 128 MHz, puis on configurait la mémoire Flash et le cache (dans les cas où il était activé). Les valeurs de ces paramètres étaient choisies de manière à assurer le bon fonctionnement de la mémoire en fonction de la fréquence d'horloge choisie ainsi qu'à améliorer, où possible, la prédictibilité du fonctionnement de la cible.

Le cœur e200z6 était configuré mode "Write Through" (pour le dernier étage du pipeline), et la prédiction de branches était désactivée. Ces choix ont été motivés par des limites d'aiT. Cependant, ces choix ne ralentissent pas beaucoup la cible tout en assurant une bonne prédictibilité du comportement du processeur.

En ce qui concerne le cache, les mesures ont été réalisées systématiquement pour le cas où il était activé et pour le cas où il était activé. Quand le cache était activé, il fonctionnait en mode séparé pour les instructions et les données.

B.4.2 Compilation du code

Dans le cadre de cette évaluation, les programmes tests ont été compilés avec l'outil Codewarrior dédié au MPC5554. La compilation était optimisée pour la vi-

tesse d'exécution et le niveau général d'optimisation était réglé au niveau 4 (renseigné ainsi : code réduit, debug difficile et compilation lente).

B.4.3 Référence : mesure sur émulateur

Pour servir de référence, le code était chargé sur une cible réelle sur laquelle était branché un émulateur permettant de surveiller l'exécution du code à l'aide du logiciel WinIDEA. Ce dispositif permet de suivre instruction par instruction l'exécution du code embarqué sur le processeur ainsi que d'enregistrer des traces ou même d'utiliser un « profiler » qui permet d'obtenir facilement les temps d'exécution maximaux, moyens et minimaux des fonctions que l'on désire étudier.

Pour la comparaison avec les calculs d'aiT, on utilise les valeurs maximales constatées en utilisant le profiler.

B.4.4 Utilisation d'aiT

En ce qui concerne l'utilisation de l'outil, il est important d'avoir une bonne connaissance du code analysé ainsi qu'une bonne connaissance de la cible utilisée. L'obtention de résultat se fait par plusieurs étapes successives de raffinement décrites brièvement ci-après.

Configuration de la cible

La première étape consiste à indiquer à l'outil la configuration du microprocesseur cible. En particulier, il est très important de donner les bons paramètres de configuration de la mémoire flash (nombre de cycles d'attentes à la lecture et modes de prefetching entre autres), et du cache. Les paramètres choisis sont montrés dans les figures B.4 et B.5 en annexe. Afin de transformer le nombre de cycles calculés par l'outil en un temps, la fréquence d'horloge de la cible doit être indiquée dans le fichier d'annotations.

Enfin, si le cache est activé, il faut préciser dans les annotations quelles adresses mémoires sont gardées en cache. Pour cette évaluation, les instructions lues dans la mémoire flash et les données stockées dans la RAM était gardées en cache.

Pour cette évaluation, cette étape n'a été nécessaire qu'une seule fois comme la configuration du MPC5554 était la même pour tous les programmes.

Résolution des conflits pour l'obtention d'un premier résultat

La seconde étape consiste à obtenir un premier résultat de pire temps d'exécution avec l'outil. Pour cela, il faut tout d'abord choisir un point d'entrée du programme pour effectuer l'analyse puis résoudre un à un les conflits de l'outil. Par exemple, il est parfois nécessaire d'indiquer manuellement des bornes pour des

boucles ou l'adresse du pointeur de pile au début de l'exécution (même si cette dernière peut être généralement devinée par l'outil).

C'est l'étape la plus importante de l'analyse et elle doit être réalisée pour chacun des différents programmes. En fonction du programme analysé, cette étape peut être aussi bien assez facile qu'assez compliquée. Les complications viennent en général de boucles dont l'outil n'arrive pas à trouver automatiquement la borne. Il revient alors à l'utilisateur d'intervenir en utilisant sa connaissance de l'algorithme ou en faisant des hypothèses.

A la fin de cette étape, l'utilisateur est en possession d'une première valeur du pire temps d'exécution. Les étapes suivantes ont pour but de donner des informations à l'outil afin d'affiner les calculs et de se rapprocher du pire temps d'exécution réel dont les calculs nous donne seulement une borne supérieure.

Paramétrage des contextes

L'étape suivante consiste à jouer sur la longueur des contextes pris en compte pendant l'analyse du code ainsi que sur le déroulage des boucles et fonction itératives. Il faut trouver un compromis entre le temps et la mémoire nécessaires aux calculs et la précision désirée.

A l'exception d'un programme comportant des boucles imbriquées, il était possible d'augmenter arbitrairement ces paramètres sans dégrader sensiblement la vitesse de l'analyse. Pour la longueur des contextes, il est recommandé de s'aligner sur la profondeur du graphe de flux de contrôle. Pour le déroulage des boucles, il est très recommandé de distinguer au moins la première itération par rapport aux autres (ce qui est le paramètre par défaut). Ensuite, il est recommandé d'augmenter progressivement le déroulage jusqu'à ce que le pire temps d'exécution se stabilise ou que l'analyse devienne trop longue.

Il d'agit d'une étape très simple à mettre en œuvre, assez rapide et permettant d'affiner le résultat dans des proportions assez légères mais non négligeables.

Zones mémoires

Ensuite, les annotations offrent la possibilité de spécifier certaines plages d'adresses mémoires comme volatiles, en lecture ou écriture-seule, ou comme jamais accédées. Grâce à ce moyen, cela permet de faire des hypothèses moins pessimistes sur les temps d'accès ou de trouver les bornes de certaines boucles. Par exemple, la spécification d'une variable en lecture seule peut permettre à l'outil d'exclure certains chemins d'exécution ou de borner certaines boucles.

Cela peut permettre d'affiner les résultats mais demande une bonne expertise du code.

Flux de contrôle

Enfin, il est également possible d'ajouter des annotations concernant le flux de contrôle. Par exemple, il est possible d'indiquer le nombre d'occurrence de certains fragments du flux de contrôle de façon relative du type : « l'instruction à l'adresse 0x.... est exécutée trois fois plus que l'instruction à l'adresse 0x... ». Ou encore, il est possible d'indiquer que certaines branches ne sont jamais empruntées.

Ceci peut améliorer peut changer considérablement les résultats obtenus mais nécessite une très bonne connaissance du programme analysé.

Valeurs d'entrée

Pour aller plus loin dans le raffinement des résultats, il est possible de restreindre explicitement les plages dans lesquelles certaines variables varient. On peut également préciser la valeur des registres de base, ce qui pourrait théoriquement permettre de fixer un contexte précis.

Cependant, les tentatives de spécifier un contexte en précisant la valeur de certains registres de base correspondant aux entrées des programmes étudiés ont été infructueuses pendant l'évaluation.

Autres

Il existe encore d'autres possibilités de raffinement qui demandent une grande expertise du code ou de la cible. Néanmoins, les possibilités les plus courantes et accessibles ont été décrites précédemment.

B.5 Résultats

B.5.1 Installation et prise en main de l'outil

Tout d'abord, aiT s'installe très facilement sous Windows via un installateur standard. L'interface est simple et agréable et ne présente aucune difficulté à prendre en main. La création d'un projet et le lancement d'une analyse se font assez aisément. L'outil de visualisation des graphes demande cependant un peu de temps à prendre en main quand on veut naviguer dans des graphes de tailles conséquentes mais l'ergonomie générale de l'outil est satisfaisante. La Figure B.2 donne une capture d'écran de l'outil.

Les difficultés rencontrées proviennent généralement de la complexité du code étudié et des méthodes d'analyse utilisées. Il est donc nécessaire d'avoir un minimum d'expertise de la cible utilisée et du code étudié. L'évaluation de l'outil a été réalisée après une formation initiale par des consultants d'AbsInt. L'expérience montre que cette formation est très utile afin d'obtenir des résultats satisfaisants avec l'outil.

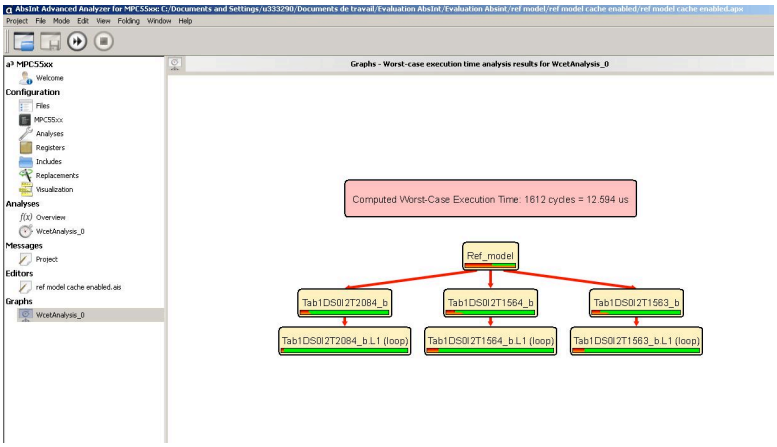

Figure B.2 – Capture d'écran de aiT.

Les résultats de cette évaluation ont été obtenus en utilisant la version 104968 de aiT pour les MPC55xx.

B.5.2 Utilisation de l'outil

A l'utilisation, l'outil révèle un degré important d'automation et ne demande de l'assistance que dans un nombre limité de cas où il n'est pas possible de continuer sans information supplémentaire. Pendant l'évaluation, l'outil a eu besoin d'assistance pour deux types de cas différents.

Le premier cas où il fallait intervenir était lié au compilateur. La connaissance du compilateur aide l'outil à mieux reconnaître certaines structures. Au moment de l'étude, le support complet du compilateur CodeWarrior n'était pas intégré. Il a fallu parfois indiquer manuellement que certaines zones étaient en écriture seule pour que l'outil détermine certains branchements. Lors de premiers essais avec le compilateur "Diabdata" de WindRiver, ces mêmes branchements étaient déterminés automatiquement par l'outil.

Le second cas où il fallait intervenir était lié à certaines boucles "while" pour lesquelles aiT n'arrivait pas à trouver le nombre maximum d'itérations. Ce cas survenait en particulier en présence de blocs de cartographie. La fonction qui permettait de parcourir la carte était générique (et donc en particulier indépendante de la taille de la carte). En conséquence, aiT n'arrivait pas toujours à trouver automatiquement le nombre maximal d'itérations. Des annotations étaient donc manuellement écrites en tenant compte de la taille de la carte utilisée pour l'exemple concerné. Dans un second temps, ces annotations ont été vérifiées via le profiler de l'émulateur.

B.5.3 Vitesse de calcul

La vitesse de calcul du pire temps d'exécution par le logiciel était raisonnable pour les exemples étudiés. L'évaluation a été réalisée à l'aide d'un ordinateur portable équipé d'un processeur Core 2 duo cadencé à 1,8 GHz et de 2Go de RAM. Le temps nécessaire aux calculs excédait rarement 15 secondes. De plus, pour les exemples issus de génération automatique de code, l'outil demandait plus de temps pour générer le graphe du flux de contrôle que pour calculer le pire temps d'exécution.

B.5.4 Mesure du pire temps d'exécution

Comme expliqué précédemment, des mesures de temps d'exécution ont été réalisées afin de pouvoir apprécier quantitativement les résultats des calculs de l'outil aiT. Cependant, il est difficile de mesurer le pire temps d'exécution d'un programme. La raison de ce problème vient de la dépendance des programmes à leurs variables d'entrées. En pratique, il est impossible de tester de façon exhaustive toutes les combinaisons d'entrées possibles. Pour y remédier, on utilise des profils de mission faisant intervenir des modes de fonctionnement critiques. Cela permet en général de se rapprocher du pire temps d'exécution réel mais il est impossible de garantir d'avoir pu l'observer.

De plus, le pire cas constaté sur un profil de mission n'est pas toujours obtenu pour le même jeu de variables d'entrées si le cache est activé ou désactivé. En effet, l'utilisation du cache permet un gain de temps conséquent dans les boucles ce qui peut provoquer des changements au niveau des valeurs relatives des temps d'exécutions mesurés.

La figure B.3 montre en exemple les mesures réalisées pour un profil de mission pour la fonction "PSA#1".

Le profiler ne nous indique que le temps maximum observé sur un profil de mission. Pour cet exemple, le pire cas mesuré retenu par le profiler pour le cache désactivé arrive à la toute fin du profil de mission (en bleu, vers le point #475). Avec le cache activé, le pire cas mesuré retenu par le profiler correspond au pic en rouge vers le point #400.

Enfin, dans le cas où le cache est activé, ce type de mesure est biaisé. Comme la fonction mesurée est lancée itérativement sur la cible, le cache contient déjà une partie des instructions ou données intervenant pendant l'exécution de la fonction. En conséquence, les temps mesurés sont généralement plus faibles que si le cache avait été vide au lancement de la fonction. Par contre, lors de changements de modes de fonctionnement, il est nécessaire de recharger le cache avec de nouvelles instructions et données ce qui se traduit par des pics sur le profil de temps d'exécution. Si les instructions devaient être rechargées en cache à chaque exécution de la tâche, les temps d'exécution seraient plus longs et il n'y aurait plus de pics aux

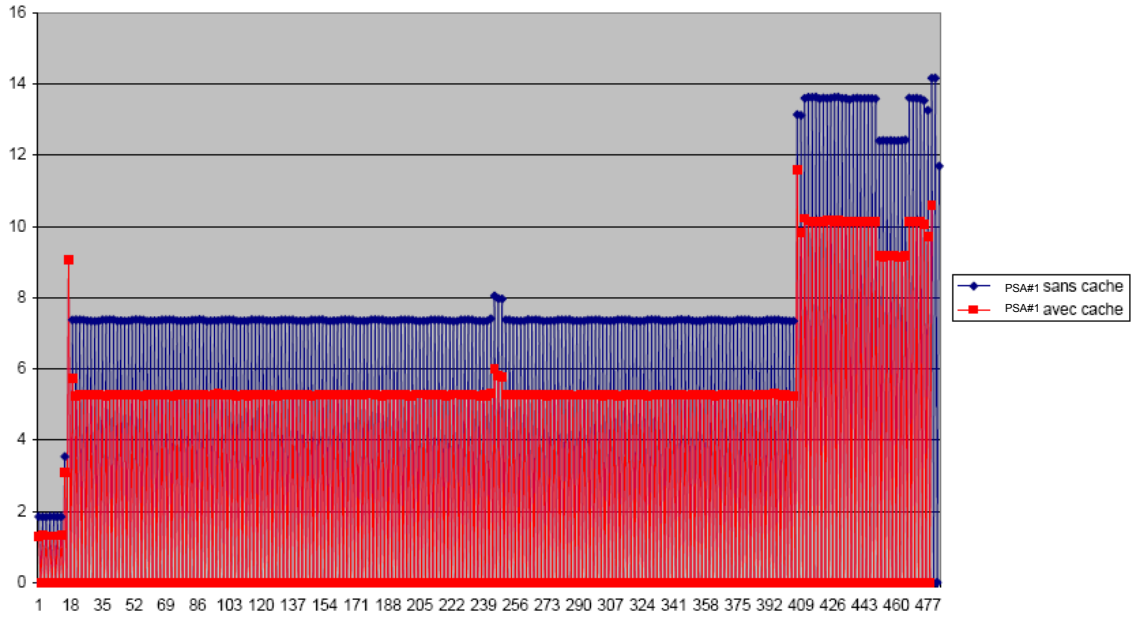

Figure B.3 – Temps d'exécution mesurés sur un profil de mission pour PSA#1.

changements de modes. Pour corriger ce biais, des mesures ont été réalisées en lançant les fonctions directement avec certains jeux d'entrées après l'initialisation de la cible. Par exemple, on essaie directement les entrées correspondant aux deux points remarqués dans la figure précédente à la première itération de PSA#1 (cas #8380 pour l'entrée causant le pic bleu, cas #8362 pour l'entrée causant le pic rouge). Comme attendu, les temps d'exécutions sont plus longs que lorsqu'on exécute le profil. Pour l'entrée #8362 de "PSA#1", le temps mesuré augmente alors de plus de 10%.

B.5.5 Chemins d'exécution

Au cours de l'évaluation, un autre constat s'est imposé : les chemins d'exécution (c'est-à-dire la séquence d'instructions exécutées) du pire cas mesuré avec l'émulateur et du pire cas trouvé par aiT ne sont que rarement identiques. Ils sont similaires pour la fonction NDES mais comportent de nombreuses différences pour les cas issus de génération automatique de code. Ces derniers cas sont caractérisés par des graphes de flux de contrôle très complexes avec de nombreux branchements conditionnels de type "if" correspondant à autant de séquence d'instructions alternatives possibles.

Ces différences de chemin d'exécution sont liées à deux phénomènes. Tout d'abord, la non-exhaustivité des profils de missions utilisés pour certains de ces

programmes fait qu'il est parfois possible de trouver un jeu d'entrée menant à un pire cas que celui mesuré lors de l'évaluation. Ce pire cas non détecté pourrait alors correspondre ou s'approcher de celui décrit par aiT. Le second phénomène est lié aux méthodes d'interprétation abstraite. Travailler avec des intervalles de valeurs peut permettre à l'outil d'emprunter certains chemins n'existant pas dans la réalité. Par exemple, il arrive que plusieurs conditions de branchements de type « if » s'excluent toujours mutuellement dans la réalité mais que l'outil emprunte quand même plusieurs d'entre eux. Cela peut arriver si des conditions de différents branchements sont validées pour plusieurs valeurs différentes comprises dans l'intervalle considéré par l'analyseur. Par exemple, considérons le code suivant :

```
If(x==1){helloWorld();}
If(x==2){helloWorld();}
...
If(x==10){helloWorld();}
```

Si l'outil analyse ce code pour un contexte où la variable "x" est dans l'intervalle [1 ;10], la procédure helloWorld() contribuera dix fois au temps d'exécution globale (au lieu d'une seule en réalité), même si x n'est pas modifié dans la procédure helloWorld().

Dans le but de pallier à la non-exhaustivité des jeux d'entrée mais aussi pour évaluer la surestimation due à l'analyse du pipeline et du cache, on contraint le flux d'exécution du programme analysé avec aiT. Pour ceci, on suit une à une les instructions exécutées pour le pire cas mesuré avec l'émulateur, puis on utilise des annotations pour exclure certaines branches du flux de contrôle pour reproduire la même séquence dans aiT.

B.5.6 Comparaison des résultats des mesures et des calculs

Du fait des remarques énoncées dans les deux sections précédentes, différentes comparaisons entre mesures et calculs ont été réalisées. Elles mettent en évidence des écarts assez variables que nous tâcherons après d'interpréter. Les résultats des mesures et des calculs sont présentés et comparés dans les figures B.6 et B.7 figurant en annexe. Le pourcentage d'écart est obtenu en divisant la différence entre les temps mesurés et calculés par le temps mesuré. Pour des questions d'échelle, les temps d'exécution mesurés pour "NDES" sont représentés avec un facteur diviseur de 30.

Pour chacune des différentes fonctions essayées, la première comparaison s'effectue entre le pire temps mesuré avec l'analyseur (en laissant tourner en boucle la fonction) et le pire temps calculé par l'outil. Les résultats sont assez variables : on observe une différence inférieure à 10% pour "NDES" sans cache et une différence

de 123% pour "Statemate" avec le cache activé. Les résultats obtenus avec "NDES" semblent montrer que l'approche utilisée pendant l'évaluation peut permettre d'obtenir de bons résultats.

Comme attendu, on remarque que les écarts sont plus forts quand le cache est activé. Pour les trois cas issus de PSA, l'écart entre les résultats est d'environ 30% avec le cache désactivé, alors qu'elle est plus proche des 50% pour le cache activé. En effet, l'analyse du cache ajoute de la complexité et les hypothèses pessimistes supplémentaires doivent être réalisées (à cause de possibles cache flushes).

Pour illustrer le biais évoqué dans la section 5.4, on remesure, directement après initialisation de la cible, le pire cas observé pour la fonction "PSA#1" avec le cache activé. Comme attendu, le temps mesuré est alors plus long. L'écart entre mesures et analyse passe donc de 51,1% à 34,2%.

L'exemple "CNT" présente des écarts plus importants par rapport aux exemples de PSA : 54,2% et 94,9%. Cependant, le rapport d'analyse indique que la plupart des adresses des lectures en mémoire n'ont pas pu être déterminées ce qui a donc entraîner aiT à faire l'hypothèse que ces accès mémoires sont réalisés dans les zones les plus lentes. La restriction des accès mémoire à la RAM via une annotation permet de réduire légèrement les écarts qui restent néanmoins conséquents (cas "CNT accès restreint à la RAM" dans les graphes des résultats).

Comme expliqué précédemment, les exemples issus de génération automatique de code (les trois exemples PSA et "Statemate") ont une structure très complexe ce qui soulève les problèmes d'exhaustivité des mesures et des limitations de l'interprétation abstraite. Les séquences d'instructions correspondant à la mesure et l'analyse comparées sont généralement différentes. Pour y remédier, des annotations sont écrites de manière à analyser la même séquence d'instruction que la mesure correspondante. Les résultats de ces comparaisons ont la dénomination « annoté » dans les graphes de résultats. Comme attendu, les écarts entre mesures et analyses baissent alors très sensiblement, en particulier pour l'exemple « Statemate ». Un écart important persiste pour "Ref Model" avec le cache activé mais cela est peut-être dû aux problèmes de mesures quand le cache est activé.

B.5.7 Interprétation des résultats des mesures et des calculs

Ces dernières comparaisons permettent d'isoler la part de surestimation due à l'analyse du pipeline car l'outil suit alors strictement la même séquence d'instructions que la mesure. Après la plupart des sauts et branchements, il est nécessaire de faire l'hypothèse d'un pipeline vide et d'un cache flush afin de couvrir les pires cas possibles. Travailler avec des valeurs précises au lieu d'intervalles de valeurs permettrait de connaître précisément le contexte et donc de s'affranchir d'utiliser ces hypothèses pessimistes et par la suite de gagner en précision. Cependant, c'est l'utilisation de ces intervalles de valeurs qui permet de rendre l'analyse réalisable

en un temps raisonnable. Les résultats obtenus semblent indiquer que l'analyse du pipeline (et donc du cache) induit de 10 à 25 % de surestimation par rapport aux mesures.

En résumé, l'écart entre les mesures et les calculs semble donc avoir trois composantes. Tout d'abord il y a l'écart entre la mesure et le pire cas d'exécution réel dû à la non-exhaustivité des mesures et la difficulté pratique de s'approcher du pire cas avec le cache activé. Ensuite, une seconde part de l'écart peut s'expliquer par des choix différents, voir non-cohérents, pendant la détermination du pire chemin d'instructions par aiT. La dernière partie de l'écart provient des hypothèses pessimistes effectuées lors de l'analyse du pipeline et du cache.

Les deux premières composantes sont difficilement quantifiables de façon générale car elles dépendent vraiment de la complexité des exemples étudiés. Il est cependant possible de limiter leur importance via une bonne connaissance du programme analysé. En effet, une bonne connaissance du programme permet d'élaborer des profils de missions pertinents et éventuellement d'utiliser des annotations pour contraindre de façon pertinente le chemin d'instruction dans aiT. La dernière composante est inévitable mais son influence semble se limiter à environ 25% d'écart.

En ce qui concerne les exemples issus de PSA, l'écart est de l'ordre de 50% avec le cache. Le calcul donne donc des résultats du même ordre de grandeur que les mesures ce qui pourrait permettre de prouver certaines propriétés de sûreté de fonctionnement. Cependant, la surestimation est trop importante pour pouvoir utiliser les résultats de l'outil pour s'attaquer à des questions de dimensionnement.

B.6 Conclusion

A notre connaissance, aiT est encore peu utilisé dans l'industrie automobile. La première raison est que cela nécessite un investissement de temps important et une bonne expertise pour obtenir des résultats satisfaisants. En effet, il faut d'abord pouvoir décrire et configurer le microprocesseur utilisé de façon correcte. Ensuite, il est nécessaire d'avoir une bonne connaissance de la fonction étudiée pour pouvoir fournir les annotations requises. Enfin, une bonne expertise de l'outil est nécessaire au raffinement des résultats. Cela nécessite donc les compétences réunies de l'automaticien qui conçoit le modèle de la fonction, de l'électronicien qui connaîtrait la cible et son paramétrage (ce qui est généralement du périmètre des fournisseurs automobiles), et d'une tierce personne connaissant les méthodes d'analyse temporelles et le fonctionnement de l'outil.

Pour l'évaluation, nous avons utilisé du code généré automatiquement à partir de modèles de type Simulink pour des fonctions électroniques embarquées dans les véhicules. La génération automatique de code permet de concevoir des fonctions de plus en plus complexes et utilisant de plus en plus de paramètres. L'analyse exhaustive des propriétés temporelles de ces programmes en est rendu d'autant

plus difficile. Les expérimentations réalisées pendant l'étude montrent que, pour de telles fonctions, il est relativement aisé d'obtenir avec aiT un pire temps d'exécution, car le code généré est bien structuré. Cependant, les pires temps d'exécution donnés par l'outil sont supérieurs de 50% par rapport aux pire temps mesurés. Ceci est dû à la complexité des modèles qui comportent un grand nombre de chemins d'exécutions alternatifs. Cet écart s'explique par la non-exhaustivité des mesures et par la possibilité des méthodes d'interprétation abstraites d'utiliser des séquences d'instructions impossibles lors d'une exécution réelle sur un microprocesseur. En général, le concepteur de la fonction est capable de produire les profils de missions mettant en évidence les cas critiques de façon à limiter le premier problème. Par contre, pour pallier au second problème, il est nécessaire de contraindre les séquences d'instructions analysées par l'outil (ce qui nécessite d'avoir une cible réelle comme référence), ou de changer d'outil. En effet, des outils comme RapiTime permettent de s'appuyer sur des mesures pour donner un pire temps d'exécution. Le problème de la non-exhaustivité des mesures subsiste alors, mais l'écart entre le pire temps calculé par l'outil et les mesures devrait être sensiblement plus faible.

La connaissance du pire temps d'exécution a principalement deux intérêts pour un constructeur automobile comme PSA : le dimensionnement de l'architecture électronique et la vérification de la sûreté de fonctionnement. Au regard des efforts nécessaires pour le calculer, cela n'a d'intérêt que pour les tâches critiques au niveau temps-réel. Pour des tâches de faible criticité, il est plus intéressant de procéder à de simples mesures du temps d'exécution qui seront plus rapides et moins coûteuses. Les écarts de résultats obtenus pendant l'étude suggèrent que l'outil aiT d'AbsInt n'est que peu adapté au dimensionnement de l'architecture électronique. La sensibilité aux entrées des fonctions utilisées dans le domaine de l'électronique embarquée dans l'automobile nécessite de se baser sur des mesures pour obtenir des pire temps d'exécutions plus fins. Par contre, dans le domaine de la sûreté de fonctionnement, il est nécessaire d'avoir une approche exhaustive par rapport aux paramètres utilisés, ce que seuls les outils d'analyse statique peuvent apporter. Pour des besoins de certification d'applications critiques du point de vue de la sûreté de fonctionnement, et en particulier en perspective de la norme ISO 2626-2, l'outil aiT d'AbsInt peut être une bonne solution pour calculer des pire temps d'exécution.

Annexes

Paramètres de configuration du microprocesseur cible pour aiT

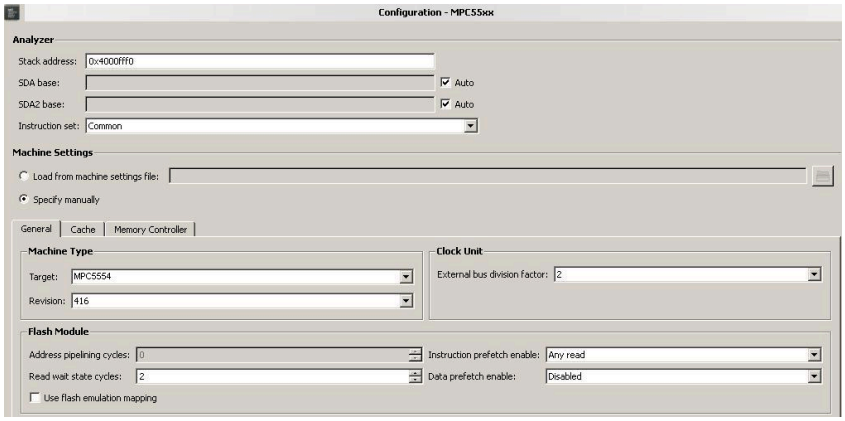

Figure B.4 – Paramètres de l'onglet "General".

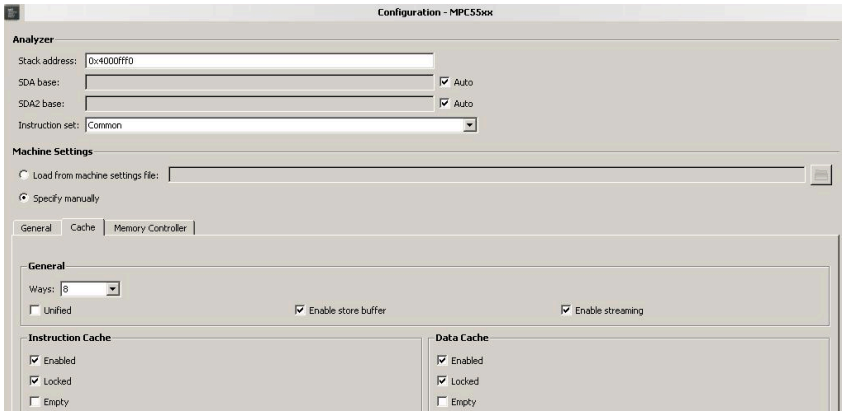

Figure B.5 – Paramètres de l'onglet "Cache".

Graphes des résultats

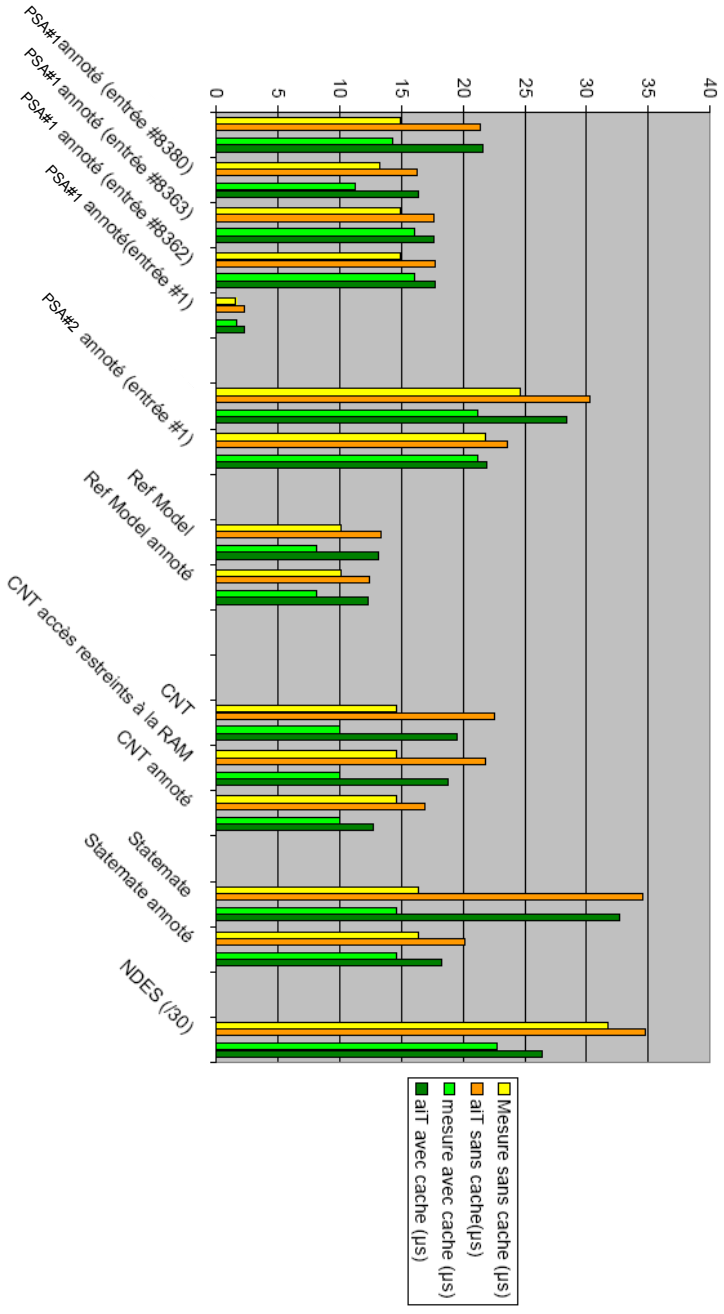

Figure B.6 – Résultat des calculs et des mesures.

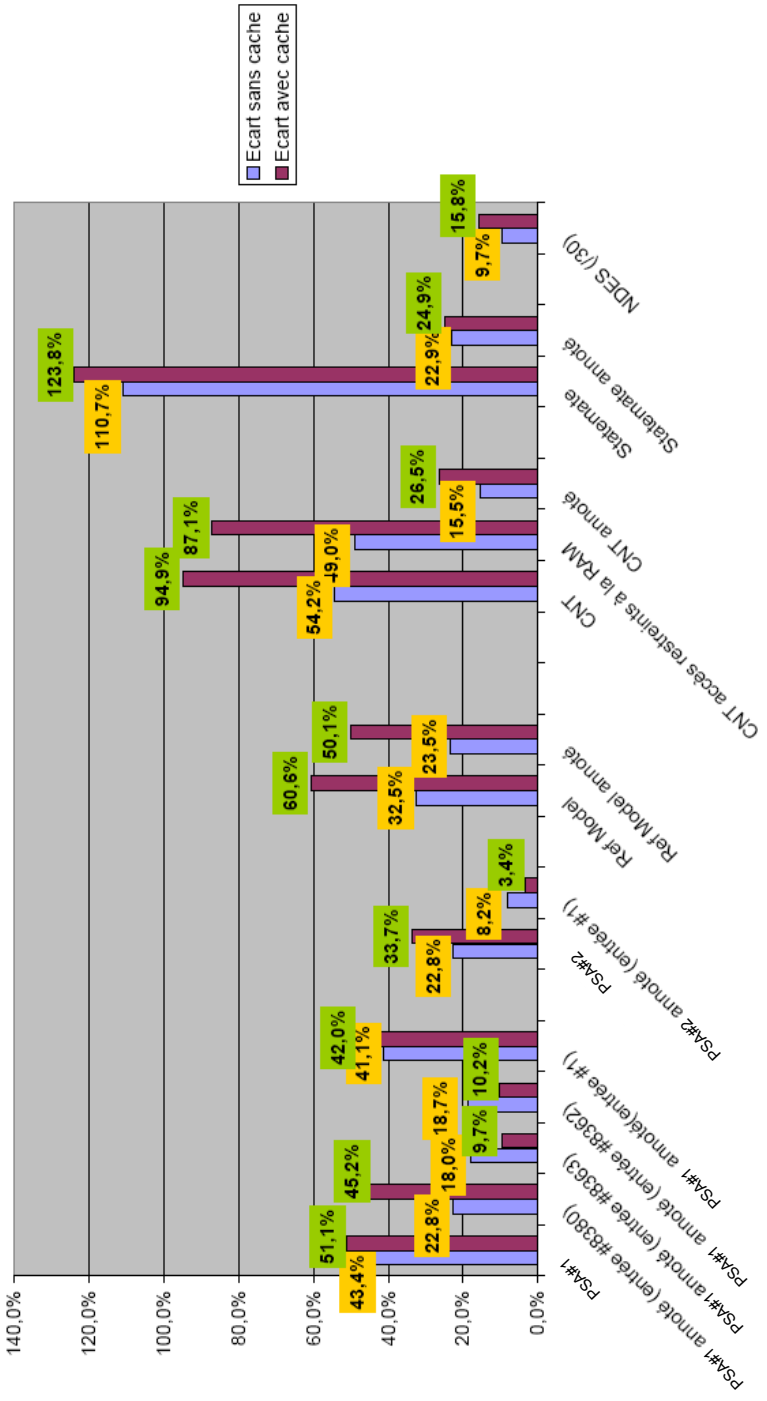

Figure B.7 – Comparaison des résultats.

Annexe C

Descriptif des brevets

Sommaire

C.1 Introduction

Nous présentons ici un descriptif des trois brevets [7, 8, 9] dérivés des travaux sur la simulation des réseaux CAN présentés dans le chapitre II. Ces trois brevets sont liés les uns avec les autres et décrivent des dispositifs intégrés aux contrôleurs de communication CAN visant à détecter et corriger les situations d'encombrement du réseau où les déphasages entre les calculateurs sont tels que les temps de réponse des trames deviennent longs. Les deux dispositifs de résolution des situations d'encombrement sont présentées ensemble dans la mesure où ils partagent une base commune.

C.1.1 Dispositifs ou procédés existants

A l'heure actuelle, il n'existe pas de dispositif ou de procédé connu qui soit embarqué dans le véhicule pour surveiller en temps réel les temps de réponse des messages émis sur les réseaux de communication. Certains boitiers électroniques communicants gardent temporairement en mémoire les dates auxquelles les messages sont émis avec succès. Cependant, comme les dates auxquelles les messages sont prêts à être envoyés (c'est-à-dire quand ils sont copiés dans les buffers d'émission) ne sont en général pas sauvegardées, les temps d'attentes ne sont pas connus en temps réel par le système en fonctionnement. Quand bien même lorsque les dates auxquelles les messages sont prêts à être envoyés sont connus, le temps de gestion par logiciel de ces dates est important et il n'existe pas de diagnostic embarqué signalant en temps réel l'état d'encombrement du bus.

C.1.2 Problématique

Lors du fonctionnement d'un véhicule, il peut se produire des situations où les temps de réponse des messages envoyés sur les réseaux s'allongent. Un seul message pouvant être transmis à la fois sur un réseau, il se produit parfois des collisions. En cas de collision, les messages sont réémis ultérieurement ce qui augmente donc leurs temps de réponse. Ces situations peuvent, dans certains cas, s'installer durablement. Il est donc important de pouvoir détecter en temps réel ces situations de surcharge afin de pouvoir y remédier.

Dans certains cas les messageries définies dans l'AEE (Architecture Electrique et Electronique) des véhicules sont intialement configurées à la mise sous-tension de manière à ne pas envoyer tous les messages en même temps. Cependant les dérives d'horloge des boitiers électroniques remettent en cause au bout d'un certain temps la pertinence de l'ordonnancement initial.

Par exemple, sur un réseau CAN, chaque calculateur est cadencé par sa propre horloge. Ces horloges ne sont pas synchronisées et sont donc sujettes à des dérives. Ces dérives s'expliquent par des dispersions de fabrication, le vieillissement des

horloges et les variations de température. Même si ces dérives sont faibles, elles ont pour conséquence de déphaser au cours du temps les instants, supposés périodiques, où les contrôleurs cherchent à émettre des trames sur le réseau (conformément à la périodicité de la tâche de communication d'AutoSar). Les décalages entre calculateurs évoluent donc en temps-réel pendant le fonctionnement du réseau.

Quand les messages des calculateurs sont idéalement décalés, les émissions sont entrelacées, et les interférences entre trames sont faibles. Quand les calculateurs se retrouvent synchronisés et qu'ils essaient d'émettre au même moment, alors les interférences sont fortes et les délais de réponses augmentent en conséquence. Dans la pratique, la dérive des horloges est souvent assez forte pour que l'on puisse passer de situations où le trafic sur le réseau est bien réparti à des situations où le réseau est encombré. En même temps, la dérive des horloges peut aussi être suffisamment faible pour que ces situations puissent gêner la réactivité du système pendant des durées significatives par rapport à la dynamique des organes pilotés. Nous pouvons donc observer parfois des situations durables où les temps de réponse des trames sont longs alors qu'un décalage des stations permettrait d'y remédier.

Ces situations sont subies et actuellement, il n'existe ni de moyen de les détecter au moment où elles surviennent ni de les corriger. Il faut donc une solution pour les détecter en temps réel dans l'objectif de pouvoir ensuite intervenir afin d'améliorer la réactivité et la qualité de service du réseau. L'objectif est donc d'embarquer des mécanisme temps-réel de détection et de résolution des situations d'encombrement dans les boitiers électroniques communicants.

C.2 Dispositif de détection d'encombrement d'un réseau de communication par analyse de temps d'attente de messages périodiques devant être transmis par un organe communicant

C.2.1 Présentation de l'invention

Nous décrivons dans cette section le brevet N° 1155112 [7] déposé le 10.06.2011. L'invention consiste en la définition d'un dispositif inclut dans un boitier électronique communiquant sous forme de logiciel et / ou de matériel, par exemple sous forme de matériel dans son contrôleur CAN, afin de détecter les situations d'encombrement des réseaux de communication, où les délais de réponses de certaines trames transmises sur le bus s'allongent. Cette invention permet de surveiller les statistiques des temps de réponse d'un ou plusieurs messages. A intervalles réguliers, des " règles d'encombrement " sont évaluées afin de détecter si le réseau est encombré.

C.2.2 Caractéristiques structurelles et fonctionnelles

La mise-en-œuvre de cette invention nécessite que le contrôleur de communication du boitier électronique ait certaines caractéristiques aux niveaux structurel et fonctionnel.

Au niveau structurel, il est d'abord nécessaire d'avoir un moyen de mesure pour dater la mise en attente et les émissions de messages. Pour cela, il est nécessaire d'avoir un compteur qui servira d'horloge. Certains fournisseurs (Freescale ou Renesas par exemple) proposent des calculateurs dont les contrôleurs CAN intègrent déjà un tel compteur nommé "Free Running Timer", généralement codé sur une longueur de 16 bits. Ce compteur est incrémenté à chaque fois qu'un bit de données passe sur le réseau. Avec 16 bits, il est possible d'observer 65536 bits ce qui équivaut au passage de presque 500 trames CAN de longueur maximale (8 octets de données utiles et pire cas de bit-stuffing). La longueur du compteur doit être suffisamment grande pour supporter les situations pire-cas. La valeur nominale peut être trouvée par analyse statique de la configuration de la messagerie. A titre indicatif, un compteur de 10 bits permet de prendre en compte le blocage d'un message par 7 autres messages de taille maximale. Il est donc recommandé d'avoir un compteur de 12 à 16 bits dans ce cas. Nous désignons ce compteur par " Horloge Réseau " et appelons cette valeur " Résolution Horloge Réseau ". Le Free Running Timer de 16 bits utilisé par exemple chez Freescale est une implémentation possible de cette horloge réseau.

Par ailleurs, il est important de pouvoir garder en mémoire la date à laquelle un message est prêt à être émis, c'est-à-dire l'instant où un message est copié dans un buffer d'émission. Certains fournisseurs (comme Freescale) intègre déjà un espace mémoire dans les buffers d'émission des contrôleurs CAN afin d'y écrire la date où la trame est émise avec succès. Cette invention nécessite la présence de champs de datation dans les buffers d'émissions. Notre invention comprend deux espaces mémoire dans le buffer d'émission du message comme iuulustré dans la Figure C.1. Dans le premier espace mémoire, l'instant retenu sera celui où le message est prêt à être envoyé. Dans un deuxième espace mémoire, nous proposons d'inscrire la date de l'instant où le message est émis avec succès (variante A) ou directement la durée d'attente calculée à la volée par différence des deux dates (variante B). Ces données sont gardées en mémoire de manière à pouvoir les collecter quand voulu, et donc éventuellement de manière asynchrone avec l'émission réussie de la trame.

Ensuite, la mise-en-œuvre de cette invention nécessite de la mémoire pour stocker les statistiques des délais de réponses. L'invention nécessite de collecter les délais de réponses des messages émis par le calculateur. Les délais de réponse des messages surveillés sont collectés sur une fenêtre glissante de N messages consécutifs. N est à définir en fonction de la criticité des fonctions et de la mémoire disponible. Plus N est grand, plus les statistiques seront fines. Pour garder en mé-

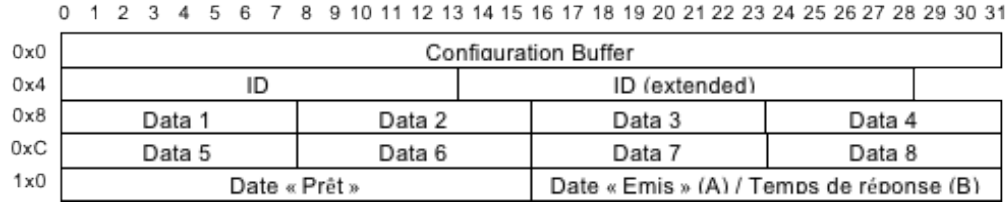

Figure C.1 – **Exemple d'organisation possible d'un buffer d'émission/réception pour un message dans le cas d'un contrôleur CAN.** Dans cet exemple, la résolution de l'horloge réseau maximale supportée est de 16 bits. La première ligne contient des informations de configuration du buffer. Les 3 lignes suivantes contiennent les informations relatives à la trame CAN à envoyer. Enfin, la dernière ligne contient les deux champs nécessaires à la mesure du temps de réponse. Le premier champ " Date prêt " sera renseigné quand la trame CAN aura fini d'être copiée dans le buffer. Le second champ est renseigné quand la trame CAN aura été émise avec succès. Dans la variante A, la valeur de l'horloge réseau y sera inscrite. Dans la variante B, le temps d'attente depuis " Date prêt " y sera inscrit.

moire ces délais de réponse, il est nécessaire d'avoir au moins N fois " Résolution Horloge Réseau " bits de disponible par message surveillé. Cette mémoire doit être localisée au niveau du contrôleur de communication pour limiter les traitements (la copie de ces données sur de la mémoire externe ralentirait le système). Cette invention nécessite donc l'ajout d'espace mémoire au niveau du contrôleur de communication.

Enfin, le contrôleur réseau doit disposer d'un dispositif permettant d'évaluer des " règles d'encombrement " qui permettent de définir une situation d'encombrement réseau. Ces règles sont définies à la conception et dépendent des applications. Elles doivent répondre " vrai " ou " faux " et être évaluables sur la seule base des temps de réponse enregistrés et de valeurs de référence. Ces règles peuvent nécessiter de calculer les minimums, maximums, moyennes et différents quantiles. Ce dispositif d'évaluation des règles d'encombrement peut être intégré soit de manière matérielle (sous forme de FPGA), soit de manière logicielle.

La figure C.2 montre un schéma de l'architecture du contrôleur réseau modifié.

Sur le plan fonctionnel, le contrôleur de communication doit pouvoir réaliser les fonctions suivantes :

- Capture de la valeur de l'Horloge Réseau à chaque instant
- Enregistrement de la valeur de l'Horloge Réseau dans l'espace réservé à cet effet dans le buffer d'émission d'un message au moment où le message correspondant a fini d'y être copié.
- Enregistrement de la valeur de l'Horloge Réseau dans l'espace réservé à cet effet dans le buffer d'émission d'un message au moment où le message cor-

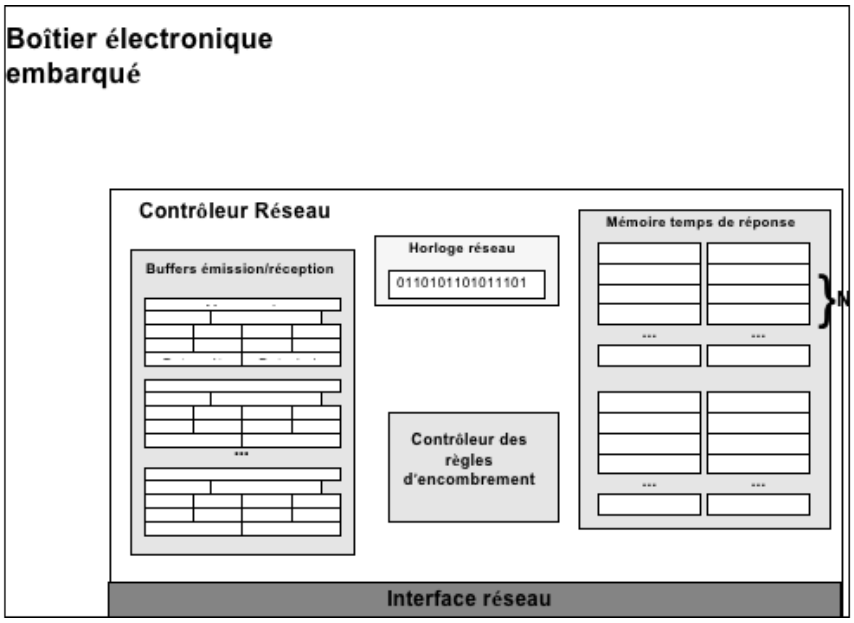

Figure C.2 – Schéma d'architecture du dispositif intégré au contrôleur réseau

respondant est émis avec succès (Variante A).

- Enregistrement du temps d'attente dans l'espace réservé à cet effet dans le buffer d'émission du message au moment où le message est émis avec succès (Variante B). Le temps d'attente est calculé par la différence de la date d'émission avec succès du message et de la date à laquelle il était prêt à être émis, modulo la valeur Résolution Horloge Réseau (car le compteur peut être revenu à Zéro s'il a atteint entretemps sa valeur maximale).
- Enregistrement du temps de réponse de la trame dans l'espace mémoire prévu à cet effet. L'espace mémoire pour enregistrer les statistiques peut être organisé comme une file FIFO de N* " Résolution Horloge Réseau " bits. Pour cela, on utilise un pointeur indiquant l'adresse mémoire pour le prochain délai à enregistrer. Ce pointeur est incrémenté de " Résolution Horloge Réseau " bit à chaque fois et revient à Zéro une fois sa valeur maximale atteinte. Le temps de réponse est soit lu directement (variante B) soit calculé à ce moment (variante A) comme fait pour la variante B avant de l'écrire dans le buffer d'émission.
- Vérification des " règles d'encombrement ". Cette vérification sera faite périodiquement (selon une période à définir), par exemple tous les N messages, une fois que le pointeur mémoire atteint retourne au début de l'espace mémoire où sont enregistrés les délais.

C.2.3 Description du fonctionnement du dispositif

Tout d'abord, cette invention nécessite un contrôleur réseau présentant les caractéristiques exposées précédemment :

- une " Horloge Réseau " de résolution suffisante,
- deux champs de la taille correspondante dans les buffers d'émissions/réception des messages pour écrire la date à laquelle un message est copié dans le buffer ainsi que la date à laquelle il est émis avec succès (variante A) ou directement le temps de réponse (variante B),
- de l'espace mémoire supplémentaire pour enregistrer N temps de réponse,
- un dispositif de contrôle des règles d'encombrement.

Ensuite, il faut configurer le contrôleur réseau par rapport aux besoins :

- choisir quels messages vont être surveillés. Il n'est pas nécessaire de surveiller l'ensemble des messages émis par le contrôleur. Plus le nombre de messages surveillés est important, plus la quantité d'espace mémoire nécessaire est importante. De plus, cela nécessite plus de puissance de calcul au niveau du contrôleur au moment de l'évaluation des règles permettant de détecter l'encombrement du trafic sur le réseau. Remarque : si les temps de réponse d'un message augmentent, cela est souvent lié à l'augmentation des temps de réponse des autres messages. C'est pour cela qu'il n'est pas nécessaire de surveiller tous les messages.
- choisir la taille des fenêtres glissantes à surveiller (choisir la valeur de N pour les messages),
- choisir la périodicité de l'étape de vérification des règles d'encombrement,
- définir les règles d'encombrement. Ces règles peuvent varier en fonction des réseaux. Une analyse préliminaire par simulation permettant de prévoir les performances du réseau peut aider à la définition de ces règles.

Au moment de l'exécution, le contrôleur enregistre dans le champ approprié la date à laquelle les messages surveillés ont fini d'être copiés dans les buffers d'émission (Figure C.3, étape A). Au moment où le message est émis avec succès, le contrôleur relève la date et écrit la date (variante A) ou le temps de réponse de ce message (variante B) dans l'espace mémoire correspondant du buffer d'émission du message (Figure C.3, étape B). Ensuite, le temps de réponse est écrit dans l'espace mémoire dédié à cet effet (Figure C.3, étape C). Ceci peut-être fait directement avec l'étape précédente ou plus tard mais toujours avant que le buffer d'émission serve à émettre ou recevoir un autre message (moment auquel il sera effacé). Dans le bloc mémoire dédié à l'enregistrement des temps de réponse, l'adresse de l'espace mémoire correspondant est donnée par un pointeur dont la valeur est mise à jour pour désigner l'adresse qui servira pour le prochain temps de réponse de ce message. Quand le pointeur prend des valeurs définies par le concepteur (à la fin d'un cycle entier, tous les messages ou autre choix), la phase de vérification des

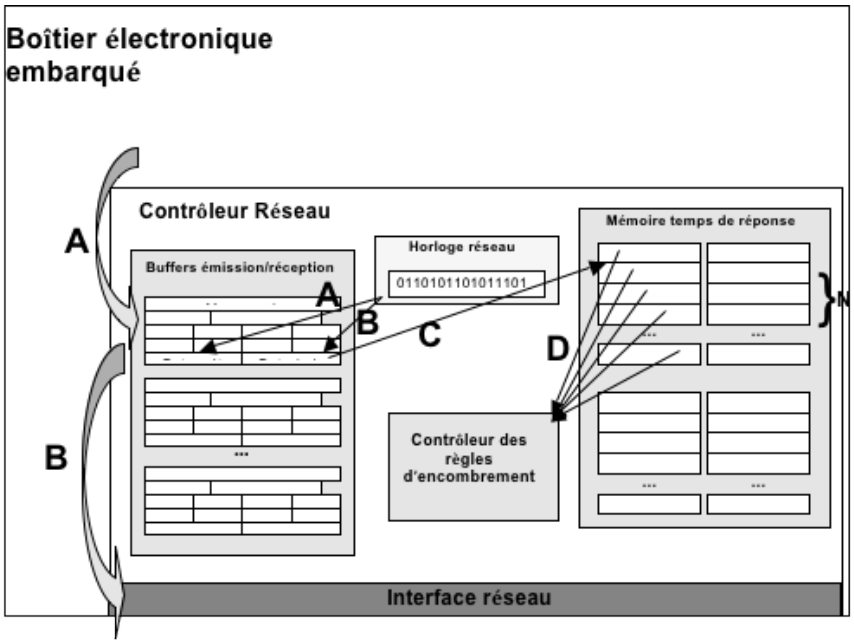

Figure C.3 – Fonctionnement du dispositif de détection des situations d'encombrement

règles d'encombrement est lancée (Figure C.3, étape D). Chaque règle est alors évaluée pour détecter si le réseau est qualifié " d'encombré ".

Par exemple, sur un réseau CAN, nous décidons de surveiller les temps d'une trame A de priorité haute et de période 20 ms ainsi qu'une trame B de priorité basse et de période 100 ms. Nous allons utiliser une fenêtre de 50 trames pour la trame A et une fenêtre de 10 trames pour la trame B. Nous décidons de vérifier toutes les secondes, ce qui correspond à la durée des fenêtres de surveillance, les règles suivantes :

Trame A :

- plus de 80% des trames ont un temps de réponse supérieur à " T80_encombrement "
- moins de 10% des trames ont un temps de réponse inférieur à " T10_encombrement "

Trame B :

- le temps de réponse minimum observé est supérieur à une valeur de référence " Tmin_encombrement "
- la différence entre le minimum et le maximum des temps de réponse est supérieure à " T_delta " et la moyenne des temps de réponse est plus proche du maximum que du minimum

Nous pourrions bien sûr observer des trames de différentes périodes, prendre une autres période d'observation, observer plus ou moins d'échantillons avant de vérifier la règle.

Ensuite nous pouvons considérer que la règle doit être validée ou invalidée respectivement plusieurs fois avant d'avoir un diagnostic positif ou négatif.

C.3 Dispositifs de résolution d'encombrement au sein d'un réseau de communucation

C.3.1 Présentation des inventions

Nous décrivons dans cette section les brevet N° 1155109 [8] et N° 1155111 [9] déposés le 10.06.2011. intitulés respectivement "Dispositif de résolution d'encombrement au sein d'un réseau de communication, par introduction d'un décalage temporel lords de la transmission d'un message, puis lors de la production des messages suivants à transmettre" et "Dispositif de résolution d'encombrement au sein d'un réseau de communucation, par recalage temporel global des instants de production de bits des message des organes communicants". Les deux brevets correpondant respectivement aux modes "local" et "global" décrit ci-après.

Les inventions concernent un dispositif qui permet de résoudre des situations d'encombrement d'un bus de communication. Il s'agit précisément des situations d'encombrements temporaires qui peuvent survenir suite aux dérives d'horloges entre des boîtiers reliés par un bus et/ou à une mauvaise synchronisation initiale d'un ou plusieurs des boîtiers électroniques reliés à ce bus de communication. Pour ce faire, ce dispositif permet de décaler dans le temps les émissions d'un ou plusieurs boîtiers, en les resynchronisant par rapport à celles des autres boîtiers.

C.3.2 Caractéristiques structurelles et fonctionnelles

Nous revendiquons deux modes de fonctionnement alternatif pour ce dispositif : un mode "local" et un mode "global", qui seront détaillés plus loin.

En mode local, une installation du dispositif est présente sur un ou éventuellement plusieurs boîtiers mais fonctionne de manière autonome par rapport aux autres boitîers. Le mode local comprend deux étapes. Dans la première étape, les messages sont retardés quand ils sont dans les buffers d'émissions afin de tester le résultat tout en ayant la possibilité de faire varier ce retard si nécessaire. La deuxième étape permet de mettre en œuvre durablement les décalages des émissions des messages tout en optimisant les temps de traitement en entrées et sorties du boîtier. Pour cela la resynchronisation sera effectuée directement au niveau du logiciel d'application.

En mode global, des installations du dispositif sont présentent sur plusieurs ou tous les boitîers communiquant sur le bus et fonctionne de manière programmée. Une fois le dispositif mis en marche, les émissions des messages sont resynchronisées sur tous les boitiers impliqués de manière à se ramener à une situation de référence connue au niveau global du bus. Cette resynchronisation est effectuée directement au niveau du logiciel d'application.

Les deux modes de fonctionnement partagent une phase finale commune. Ils permettent de résoudre tous les deux le même problème. Le mode global est plus efficace mais demande une modification de plusieurs boîtiers, contrairement au mode local qui peut s'appliquer à un seul boîtier. La mise-en-œuvre de cette invention dans un boîtier nécessite que celui-ci ait certaines caractéristiques aux niveaux structurel et fonctionnel, selon le mode de fonctionnement choisi.

Selon ses modes de fonctionnement le dispositif peut être appliqué à un seul boîtier ou à plusieurs boîtiers. Il se situe au niveau du matériel, dans le sicilicium d'un composant électronique et/ou au niveau du logiciel du boîtier.

Il n'y ni de limite concernant le nombre de boîtiers pouvant implémenter cette invention sur un bus ni la nécessité que tous les boîtiers émettant sur ce bus soient équipés de cette invention.

Caractéristiques requises communes aux deux modes de fonctionnement

Dans un boîtier électronique il y a une tâche logiciel qui prépare les messages à émettre et qui fait les demandes de communications que nous nommerons "Tâche de communication". D'autre pas il y a un sous-ensemble matériel ou logiciel qui est en charge d'effectuer et contrôler le déroulement des communications : nous le nommerons "Contrôleur de communication". Enfin, le "détecteur de l'encombrement réseau" est un dispositif permettant de détecter une situation d'encombrement réseau est un pré-requis à la mise en œuvre de cette invention. Cette invention peut par exemple utiliser un premier brevet "Dispositif de détection d'encombrement d'un réseau de communication par analyse de temps d'attente de messages périodiques devant être transmis par un organe communicant". Le nouveau dispositif présenté dans cette invention peut donc se combiner avantageusement au brevet N° 1155112 [7].

De plus, il est aussi requis que "détecteur de l'encombrement réseau" soit en mesure d'isoler la cause de l'encombrement réseau. En effet, il est important de discriminer les cas de défaillances comme par exemple un "babbling idiot" du cas d'une mauvaise synchronisation entre les émissions des calculateurs sur le réseau qui seul est concerné par la présente proposition d'invention.

Sur le plan fonctionnel, le "détecteur de l'encombrement réseau" doit notamment pouvoir réaliser les actions suivantes :

– Détecter et signaler une situation d'encombrement réseau dû à des tentatives

d'accès simultanés de plusieurs contrôleurs sur le réseau
- Détecter et signaler le désengorgement du réseau (i.e. la résolution du problème) après la détection d'une situation d'encombrement réseau

Rappel : ces fonctions sont un pré-requis mais leur détail n'est pas l'objet de cette invention.

Sur le plan fonctionnel, le logiciel du boîtier électronique doit notamment pouvoir réaliser les actions suivantes :
- Demander régulièrement auprès du contrôleur de l'encombrement réseau si une situation d'engorgement réseau à été résolue
- Décaler d'une durée donnée les émissions des messages du boîtier électronique, par exemple en décalant l'activation de la tâche de communication en charge de préparer et transmettre les messages au contrôleur de communication des tâches.

NB : c'est cette dernière action qui est la finalité de cette invention pour resynchroniser le boîtier électronique équipé du dispositif par rapport aux autres boîtiers électroniques.

Caractéristiques requises pour le mode de fonctionnement local

Pour le mode de fonctionnement " local ", il est requis d'avoir un moyen de mesure du temps au niveau du contrôleur de communication. Pour cela, il est nécessaire d'avoir un compteur qui servira d'horloge. Certains fournisseurs (Freescale ou Renesas par exemple) proposent des composants électroniques de type micro-processeur dont les contrôleurs de communications CAN intègrent déjà un tel compteur nommé " Free Running Timer ", généralement codé sur une longueur de 16 bits. Ce compteur est incrémenté à chaque fois qu'un bit de données passe sur le réseau. Nous désignons ce compteur par " Horloge Réseau " et appelons la durée élémentaire qui s'écoule lors d'une incrémentation " Résolution Horloge Réseau ". Le Free Running Timer de 16 bits utilisé par exemple chez Freescale est une implémentation possible de cette horloge réseau.

Par ailleurs, cette invention nécessite la présence de champs de datation de la demande d'émission faite par le contrôleur de communication. Il est important de pouvoir garder en mémoire la date à laquelle un message est initialement prêt à être envoyé. Certains fournisseurs (comme Freescale) intègrent déjà un espace mémoire dans les buffers d'émission des contrôleurs CAN afin d'y écrire la date où la trame est émise avec succès.. Notre invention comprend deux espaces mémoire dans le buffer d'émission du message comme illustré dans la Figure C.4. Dans le premier espace mémoire Date " Prêt ", l'instant retenu sera celui où le message est copié dans le buffer, c'est-à-dire la date où il pourrait en principe être émis. Dans un autre espace Date " Envoi autorisé " sera inscrite la date à partir de laquelle le message sera réellement autorisé à être émis sur le réseau en prenant

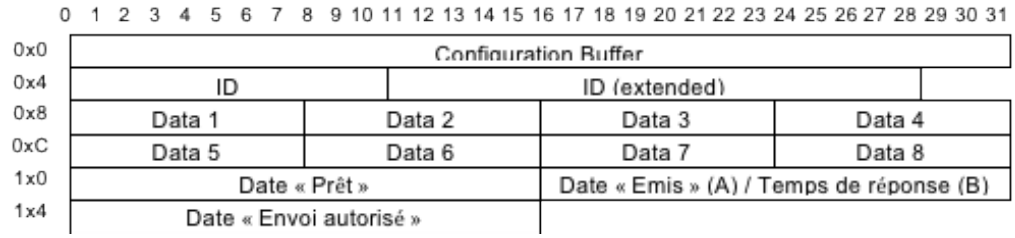

Figure C.4 – **Exemple d'organisation possible d'un buffer d'émission/réception pour un message dans le cas d'un contrôleur CAN.** Dans cet exemple, la résolution de l'horloge réseau maximale supportée est de 16 bits. La première ligne contient des informations de configuration du buffer. Les 3 lignes suivantes contiennent les informations relatives à la trame CAN à envoyer. Enfin, les dernières lignes contiennent les deux champs nécessaires à la mesure du temps de réponse. Le premier champ Date " Prêt " sera renseigné quand la trame CAN aura fini d'être copiée dans le buffer. Le second champ Date " Envoi autorisé " est renseigné sur le champ en ajoutant à " Date prêt " la valeur du retard " T_Retard ". Le champ date " Emis " / Temps de réponse est indiqué ici à titre d'exemple dans le cadre d'une combinaison avec la proposition d'invention ref. 45787 qui permet la détection des situations d'encombrement réseau.

en compte l'introduction d'un retard artificiel " T_Retard ". Nous définissons " T_Retard " comme la valeur du retard appliqué à tous les messages qui seront mis en attente dans les buffers à l'activation du dispositif. Ce retard permet de décaler les émissions du contrôleur équipé du dispositif par rapport à celles des autres contrôleurs émettant sur le réseau. Les variations et la stabilisation de cette variable seront expliquées dans la partie suivante décrivant le fonctionnement du dispositif.

Enfin, le contrôleur réseau doit disposer d'un dispositif responsable de la gestion du mécanisme de retard. Ce " contrôleur des retards des messages " peut être intégré soit de manière matérielle (sous forme de FPGA), soit de manière logicielle. Ce dispositif est en charge de fonctionnement de l'invention au niveau du contrôleur réseau. Il est en charge de communiquer avec le dispositif de détection de l'encombrement et la tâche logicielle en charge de la communication ainsi que de la gestion de la variable " T_Retard ".

Enfin, le contrôleur réseau doit disposer d'un dispositif responsable de la gestion du mécanisme de retard. Ce " contrôleur des retards des messages " peut être intégré soit de manière matérielle (sous forme de FPGA), soit de manière logicielle. Ce dispositif est en charge de fonctionnement de l'invention au niveau du contrôleur réseau. Il est en charge de communiquer avec le dispositif de détection de l'encombrement et la tâche logicielle en charge de la communication ainsi que de la gestion

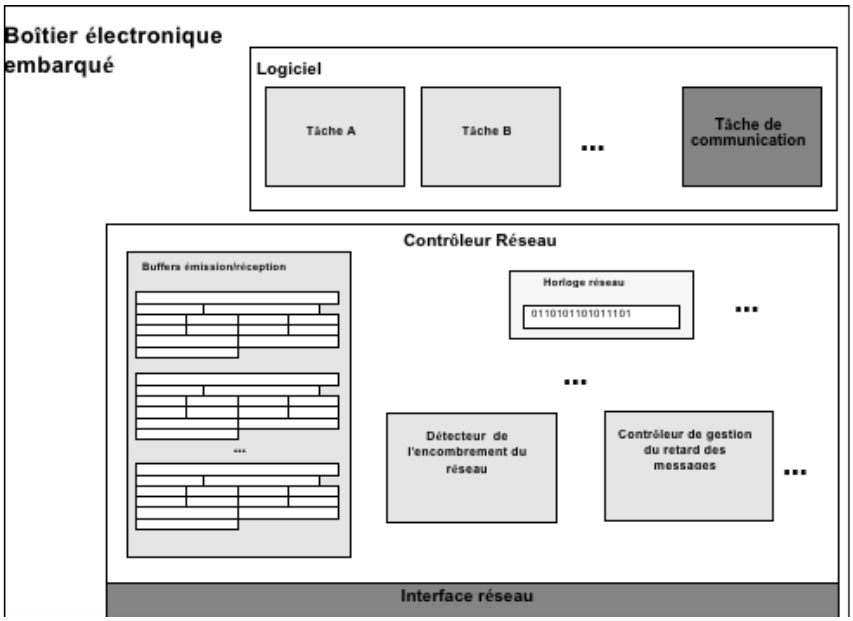

Figure C.5 – Schéma d'architecture du dispositif intégré au contrôleur réseau

de la variable " T_Retard ".

L'architecture d'un boîtier électronique équipé d'un tel dispositif est montré en Figure C.5.

Sur le plan fonctionnel, le contrôleur de communication doit pouvoir réaliser les actions suivantes :

- Capture de la valeur de l'Horloge Réseau à tout instant,
- Enregistrement de la valeur de l'Horloge Réseau dans l'espace mémoire Date " Prêt " dans le buffer d'émission d'un message au moment, quand le message correspondant a fini d'y être copié,
- Enregistrement de la valeur de la date à laquelle le message est autorisé à être émis sur le réseau après retard dans l'espace mémoire date " Envoi autorisé " dans le buffer d'émission d'un message.

Sur le plan fonctionnel, le contrôleur de gestion du retard des messages doit pouvoir réaliser les actions suivantes :

- Déterminer une valeur de " T_Retard " et la mettre à jour tant que la situation d'encombrement n'est pas résolue

Caractéristiques requises pour le mode de fonctionnement global

Dans le cas d'un fonctionnement en mode global, un message d'activation du dispositif doit être défini au niveau du réseau étudié. En cas de détection d'une

situation d'encombrement réseau, ce message sera émis sur le réseau afin d'activer simultanément ce dispositif sur tous les boîtiers qui en sont équipés.

Une situation de référence pour la tâche de communication doit être connue au niveau du logiciel du boîtier électronique. Cette situation est calculée par une analyse globale du système à la conception et garantie un décalage satisfaisant des émissions de tous les boîtiers électroniques communiquant sur le réseau. Elle définit l'ordonnancement de la tâche de communication et donc les envois des messages à partir d'une date de référence. Ainsi, quand le dispositif sera activé, les émissions des messages sont resynchronisées au niveau de tous les calculateurs équipés du dispositif et communiquant sur le réseau.

Sur le plan fonctionnel, le contrôleur de communication doit pouvoir réaliser les actions suivantes :

– Emettre un message d'activation du dispositif sur le réseau

C.3.3 Description du fonctionnement du dispositif

Modes de fonctionnement

Nous revendiquons deux modes de fonctionnement alternatif pour ce dispositif : un mode " local " et un mode " global ".

Dans le mode " local ", le dispositif fonctionne de manière indépendante et cherche de façon autonome une valeur pour " T_Retard " qui permettra de décaler les émissions des messages de manière à résoudre le problème. Dans ce cas de fonctionnement en mode " local ", afin d'éviter les interférences entre plusieurs dispositifs similaires sur le même réseau, il est préférable (mais non nécessaire) de n'équiper qu'un seul calculateur (par exemple celui qui émet le plus de messages) de ce dispositif ou alors instaurer un tour de rôle entre les calculateurs pour qu'ils se resynchronisent pas tous en même temps, par exemple par la circulation d'un jeton.

Dans le mode " global ", le dispositif fonctionne de concert avec d'autres dispositifs identiques équipant les autres boîtiers électroniques. Dans ce mode, une analyse hors-ligne doit être faite a priori au niveau du réseau global dans le but de déterminer une situation initiale satisfaisante entre tous les boîtiers électroniques en ce qui concerne l'émission de leurs messages respectifs sur le réseau. En cas de détection d'un encombrement réseau, un message est émis sur le réseau pour signaler aux boîtiers électroniques de revenir dans cette situation initiale connue. La durée à utiliser pour décaler la tâche de communication est alors calculée au niveau de chaque boîtier électronique de manière à se ramener dans cette situation de référence garantissant un déphasage satisfaisant entre eux.

Fonctionnement en mode local

En mode local, le fonctionnement du dispositif de résolution d'encombrement réseau se décompose en deux étapes :

- une première étape se déroulant uniquement au niveau du contrôleur de communication qui consiste à retarder les émissions des messages quand ils sont dans les buffers d'émission. Cette étape dure tant que la situation d'encombrement n'a pas été résolue et implique éventuellement des variations de la valeur du retard " T_Retard " infligé aux messages dans les buffers d'émission.
- une deuxième étape qui consiste à transférer le retard au niveau logiciel. Le décalage " T_Retard " est mis en place au niveau de l'ordonnancement de la tâche en charge de préparer les messages et les transférer au contrôleur de communication.

La première étape est une étape transitoire qui consiste à trouver une valeur de " T_Retard " permettant de résoudre la situation d'engorgement. Pendant cette phase, un délai supplémentaire est introduit au niveau des données transmises dans les messages alors qu'elles sont parfois rafraîchies au même moment, en dehors du contrôleur de communication (au niveau logiciel). La deuxième étape permet de résoudre ce problème. Une fois, la situation stabilisée, on décale plutôt la tâche de communication qui est responsable de la préparation des messages de manière à émettre les données les plus fraîches possibles.

Première étape de fonctionnement : retard au niveau de contrôleur de communication Pendant la première étape, à chaque fois qu'un message est copié dans un buffer d'émission (Figure C.6, étape A), la date est inscrite dans l'espace mémoire approprié dans le buffer correspondant (Figure C.6, étape B). Immédiatement, la date " Envoi Autorisé " à laquelle le message sera autorisé à être envoyé sur le réseau est inscrite dans l'espace mémoire approprié dans le buffer correspondant (Figure C.6, étape C). Cette date est obtenue en ajoutant la valeur courante de la variable " T_Retard " à la date courante donnée par l'horloge réseau (inscrite dans le buffer pendant l'étape B). Une fois que l'horloge réseau a dépassé la date " Envoi autorisé ", le contrôleur réseau peut commencer à essayer d'émettre le message sur le réseau (Figure C.6, étape D).

Cette première phase débute quand le contrôleur de l'encombrement du réseau détecte une situation d'engorgement et le signale au contrôleur de gestion du retard des messages qui calcule " T_Retard ". Tant que le premier contrôleur continue de détecter une situation d'encombrement réseau, il le signale au contrôleur de gestion du retard des messages qui fait alors varier " T_Retard " dans le but de résoudre cette situation (Figure C.6, étape E). Dès que la situation est résolue, la seconde étape est déclenchée.

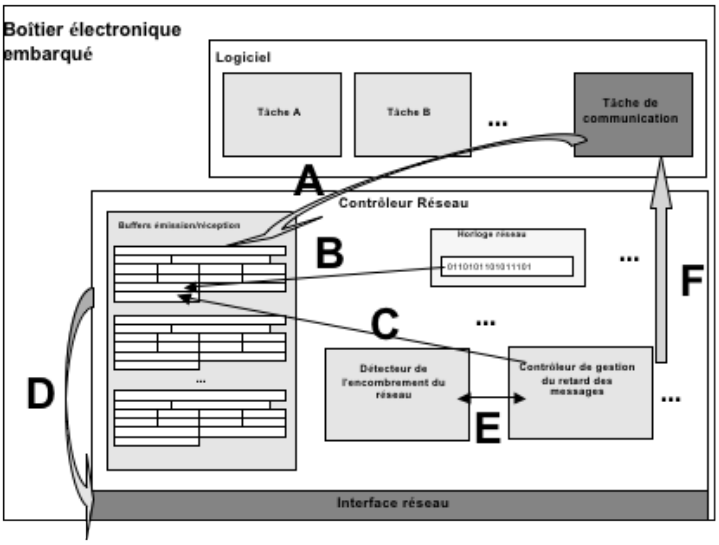

Figure C.6 – Fonctionnement du dispositif de retard des messages selon un exemple d'implémentation

Choix du retard En fonctionnement, l'état d'encombrement du réseau est régulièrement évalué par le " contrôleur de l'encombrement réseau ". En mode local, tant que le réseau est encombré, le " contrôleur de gestion du retard des messages " est alerté et met à jour la valeur de " T_Retard ". La façon de faire varier " T_Retard " dépend des cas d'applications qui peuvent avoir des objectifs de performances différents.

Une première possibilité consiste à choisir aléatoirement " T_Retard " comme dans le cas des retransmissions sur les réseaux CSMA/CD ce qui augmente l'équité dans l'accès au bus pour les messages de même importance. On peut de plus faire ce choix dans un intervalle min et max prédéfini.

Une deuxième possibilité consiste à incrémenter la variable " T_Retard " d'une valeur " Incrément_Retard ". " T_Retard " est initialisée à 0 au réveil du contrôleur et est incrémenté à chaque fois qu'un encombrement est détecté pour essayer de s'en dégager. Cependant, on veut éviter d'augmenter indéfiniment le retard des messages émis car cela augmenterait indéfiniment leurs temps de réponses et nuirait à la réactivité générale du système.. Pour cela, il est recommandé de définir une valeur seuil " Retard_Max ". Si cette valeur est atteinte, il est recommandé de changer de stratégie de choix de " T_Retard ".

De plus, il est envisageable d'utiliser d'autres approches pour choisir " T_Retard " comme par exemple l'algorithme de temporisation utilisé pour le protocole TCP. Une idée similaire peut consister à compter le nombre de fois ou il faut mettre à jour " T_Retard " avant d'avoir une résolution de l'encombrement du réseau. Si

ce nombre de fois dépasse un seuil, on multiplie par 2 (ou par un facteur prédéfini à l'avance) le retard à appliquer. Si la situation d'encombrement est résolue, on divise par 2 (ou par un facteur prédéfini à l'avance) le retard à appliquer. Ainsi, on obtient un mécanisme adaptatif qui s'ajuste au retard juste nécessaire pour sortir des situations d'encombrement. L'avantage d'un tel système est d'éviter de retarder l'information qui circule plus longtemps que nécessaire.

Deuxième étape de fonctionnement : transfert du retard au niveau logiciel
Lorsque le dispositif a trouvé une valeur de " T_Retard " permettant de résoudre la situation d'encombrement, ce retard est transféré au niveau du logiciel du boîtier électronique. La seconde étape consiste à stopper le mécanisme de retard des messages dans les buffers d'émissions et décaler les émissions des messages au niveau du logiciel. Pour cela la valeur de " T_Retard " est transmise au niveau du logiciel dont l'exécution est re-ordonnancée de manière à retarder l'exécution de toutes les instances de la tâche de communication de cette durée (Figure C.6, étape F). Ainsi, les émissions des messages sont décalées de " T_Retard " mais ceux-ci n'attendent plus au niveau du contrôleur de communication de façon à ne pas compromettre la fraîcheur des données transmises et à ne pas occuper de la mémoire au niveau du contrôleur de communication.

Fonctionnement en mode global

De manière identique au mode local, le fonctionnement du dispositif de résolution d'encombrement réseau du mode global se décompose en deux étapes :

- une première étape qui consiste à émettre ou détecter sur le réseau le message d'activation du mécanisme de résolution de situation d'encombrement réseau dans le cas de la détection d'une situation d'encombrement réseau.
- une deuxième étape qui consiste à décaler l'ordonnancement de la tâche de communication au niveau du logiciel de manière à revenir à la situation de référence déterminée à l'avance avec comme date de référence l'instant où le message d'activation du dispositif a été reçu avec succès.

Pendant la première étape, la détection d'une situation d'encombrement réseau entraîne l'émission du message d'activation du mécanisme par le boîtier électronique l'ayant détectée. Au moment où le message est émis avec succès, la seconde étape du dispositif est alors déclenchée sur tous les boîtiers équipés de cette invention (y compris celui ayant émis le message d'activation).

Chaque station équipée de ce dispositif a connaissance d'une séquence de dates d'activations de référence de sa tâche logicielle de communication par rapport à une date d'origine. La seconde étape consiste à utiliser l'instant de réception du message d'activation du dispositif comme date d'origine pour les dates d'activations des prochaines activations de la tâche logicielle. Le nouveau séquencement de la tâche

logicielle est utilisé dès que possible, en fonction du système d'exploitation du boîtier et de l'ordonnancement du reste du logiciel. Une fois la nouvelle configuration mis en route dans tous les boîtiers, les émissions des messages sont resynchronisées au niveau du réseau global afin de revenir à un fonctionnement nominal du réseau.

Par rapport au mode local, ce mode est plus rapide et promet de meilleures performances mais nécessite une évolution du logiciel dans tous les boîtiers électroniques.

C.4 Conclusion

Ces inventions sont applicables à une grande variété de bus de communication, c'est pour cette raison que les descriptions ne se limitent pas qu'au cas des contrôleurs CAN même s'ils se prêtent particulièrement bien à une application de ces brevets.

Dans des boîtiers électroniques élaborés, il est éventuellement possible de mettre en œuvre des dispositifs similaires de manière entièrement logicielle, en dehors du contrôleur de communication. Cependant, une telle solution serait moins précise et ajouterait de la charge de calcul au niveau du processeur.

Du point de vue technique et économique, ces inventions permettent de diminuer généralement les temps de réponse des fonctions pilotées par l'électronique embarquée dans le véhicule. Ceci apportera une meilleure robustesse et une meilleure réactivité aux fonctions électroniques du véhicule permettant des gains au niveau de la sûreté (pour les fonctions critiques) et également au niveau de la prestation pour le client (pour les fonctions de confort).

Annexe D

Publications et brevets

D.1 Liste des publications

D.1.1 Conférences

- [51] N. Navet, A. Monot, J. Migge, "Frame latency evaluation : when simulation and analysis alone are not enough", 8th IEEE International Workshop on Factory Communication Systems (WFCS2010), Industry Day, May 19, 2010.
- [55]N. Navet, A. Monot, B. Bavoux, F. Simonot-Lion, "Multi-source and multicore automotive ECUs – OS protection mechanisms and scheduling", invited paper at IEEE International Symposium on Industrial Electronics (ISIE 2010), Bari, Italy, July 4-7, 2010.
- [47] A. Monot, N. Navet, B. Bavoux, F. Simonot-Lion, "Multicore scheduling in automotive ECUs", Embedded Real-Time Software and Systems (ERTS 2010), Toulouse, France, May 19-21, 2010.
- [45] A. Monot, N. Navet, B. Bavoux, "Impact of clock drifts on CAN frames response time distributions", Industry Practice à ETFA'2011 - IEEE International Conference on Emerging Technology & Factory Automation,Toulouse, France, Septembre 5-9 2011
- [46] A. Monot, N. Navet, B. Bavoux, C. Maxim "Fine-grained Simulation in the Design of Automotive Communication Systems ", Embedded Real-Time Software and Systems (ERTS 2012), Toulouse, France, February 1-3, 2012.

D.1.2 Journal

- [5] A. Monot, N. Navet, B. Bavoux, F. Simonot-Lion, Multi-source software on multicore automotive ECUs - Combining runnable sequencing with task scheduling, IEEE Transaction on industrial electronics

D.2 Liste des brevets

[8] Demande de brevet : N° 1155109 déposée le 10.06.2011.
- Dispositif de résolution d'encombrement au sein d'un réseau de communication, par introduction d'un décalage temporel lords de la transmission d'un message, puis lors de la production des messages suivants à transmettre

[9] Demande de brevet : N° 1155111 déposée le 10.06.2011.
- Dispositif de résolution d'encombrement au sein d'un réseau de communication, par recalage temporel global des instants de production de bits des message des organes communicants

[7] Demande de brevet : N° 1155112 déposée le 10.06.2011.
- Dispositif de détection d'encombrement d'un réseau de communication, par analyse de temps d'attente de messages périodiques devant être transmis par un organe communicant

Bibliographie

[1] AutoSAR consortium web page. `http://www.autosar.org/`.

[2] TIMMO-2-USE project web page. `http://timmo-2-use.org`.

[3] ISO26262 Véhicules routiers - Sécurité fonctionnelle - Partie 6 : Développement du produit au niveau du logiciel, 2011.

[4] "Class C Application Requirement Considerations", SAE Technical Report J2056/1, June 1993.

[5] B. Bavoux F. Simonot-Lion A. Monot, N. Navet. Multi-source software on multicore automotive ECUs - Combining runnable sequencing with task scheduling. *IEEE Transaction on industrial electronics*, 59 :3934 – 3942, october 2012.

[6] M. Abramowitz and I.A. Stegun. *Handbook of Mathematical Functions*. Dover Publications (ISBN 0-486-61272-4), 1970.

[7] Bavoux A.Monot, B and N. Navet. Dispositif de détection d'encombrement d'un réseau de communication, par analyse de temps d'attente de messages périodiques devant être transmis par un organe communicant. Brevet n°1155112 déposé le 10.06.2011.

[8] Bavoux A.Monot, B and N. Navet. Dispositif de résolution d'encombrement au sein d'un réseau de communication, par introduction d'un décalage temporel lords de la transmission d'un message, puis lors de la production des messages suivants à transmettre. Brevet n°1155109 déposé le 10.06.2011.

[9] Bavoux A.Monot, B and N. Navet. Dispositif de résolution d'encombrement au sein d'un réseau de communication, par recalage temporel global des instants de production de bits des message des organes communicants. Brevet n°1155111 déposé le 10.06.2011.

[10] C. Aussagues and D. Roux. An OS for multicore embedded systems compliant with automotive safety standards. In *IAEC'09*, 2009.

[11] AUTOSAR Consortium. Specification of multi-core OS architecture v1.0. AUTOSAR Release 4.0, 2009.

[12] S. Baruah, D. Chen, and A. Mok. Static-priority scheduling of multiframe tasks. pages 38–45, 1999.

[13] A. Burchard, J. Liebeherr, Y. Oh, and S. H. Son. New strategies for assigning real-time tasks to multiprocessor systems. *IEEE Transactions on Computers*, 44(12) :1429–1442, Dec. 1995.

[14] Liming Chen and A. Avizienis. N-Version Proggraming : a fault tolerance approach to reliability of software operation. In *Symposium on Fault-Tolerant Computing - FTCS*, volume ChAv95, 1995.

[15] R. L. Cruz. A calculus for network delay. i. network elements in isolation. *IEEE Trans. Inf. Theor.*, 37(1) :114–131, September 2006.

[16] R. L. Cruz. A calculus for network delay. ii. network analysis. *IEEE Trans. Inf. Theor.*, 37(1) :132–141, September 2006.

[17] R.I. Davis, S. Kollmann, V. Pollex, and F. Slomka. Controller area network (can) schedulability analysis with fifo queues. In *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*, pages 45 –56, july 2011.

[18] Robert I. Davis, Alan Burns, Reinder J. Bril, and Johan J. Lukkien. Controller Area Network (CAN) schedulability analysis : Refuted, revisited and revised. *Real-Time Systems*, 35(3) :239–272, 2007.

[19] M. De Biasi, C. Snickars, K. Landernas, and A. Isaksson. Simulation of process control with WirelessHART networks subject to clock drift. In *32nd Annual IEEE International Computer Software and Applications (COMPSAC '08)*, pages 1355 –1360, 2008.

[20] J.L. Diaz, D.F. Garcia, Kanghee Kim, Chang-Gun Lee, L. Lo Bello, J.M. Lopez, Sang Lyul Min, and O. Mirabella. Stochastic analysis of periodic real-time systems. In *Real-Time Systems Symposium, 2002. RTSS 2002. 23rd IEEE*, pages 289 – 300, 2002.

[21] Mark K. Gardner. *Probabilistic Analysis And Scheduling Of Critical Soft Real-Time Systems*. PhD thesis, 1999.

[22] J. Goossens. Scheduling of offset free systems. *Real-Time Systems*, 24(2) :239–258, March 2003.

[23] J. Goossens and R. Devillers. The non-optimality of the monotonic priority assignmentsfor hard real-time offset free systems. *Real-Time Syst.*, 13(2) :107–126, September 1997.

[24] M. Grenier, L. Havet, and N. Navet. Pushing the limits of CAN - scheduling frames with offsets provides a major performance boost. In *European Congress of Embedded Real-Time Software (ERTS 2008)*, 2008.

[25] J. Kostelezky K. Barbehön-F. El-Dwaik L. Hochmuth H. Kellerman, G. Nemeth. BMW 7 Series architecture. Technical report, ATZextra, November 2008.

[26] Rafik Henia, Arne Hamann, Marek Jersak, Razvan Racu, Kai Richter, and Rolf Ernst. System level performance analysis - the symta/s approach. In *IEE Proceedings Computers and Digital Techniques*, 2005.

[27] I. Lee I. Shin. Compositional real-time scheduling framework with periodic model. *ACM Transactions on Embedded Computing Systems*, 2008.

[28] Inchron. chronSIM : a Real-Time simulator. available at `http://www.inchron.com/chronsim.html`.

[29] Inchron. Inchron Tool-Suite. available at `http://www.inchron.com/tool-suite.html`.

[30] F. Larsson P. Pettersson-W. Yi J. Bengtsson, K.G. Larsen. Uppaal - a tool suite for automatic verification of real-time systems. In *4th DIMACS Workshop on Verification and Control of Hybrid Systems*, 1995.

[31] J. Clark K. Tindell. Holistic schedulability analysis for distributed hard real-time systems. *Microprocessors & Microprogramming*, pages 117–134, 1994.

[32] A. Karrenbauer and T. Rothvoss. An Average-Case Analysis for Rate-Monotonic Multiprocessor Real-time Scheduling. In *17th Annual European Symposium on Algorithms (ESA)*, 2009.

[33] D.A. Khan, R.J. Bril, and N. Navet. Integrating hardware limitations in CAN schedulability analysis. In *8th IEEE International Workshop on Factory Communication Systems (WFCS2010)*, pages 207 –210, 2010.

[34] D.A. Khan, N. Navet, B. Bavoux, and J. Migge. Aperiodic traffic in response time analyses with adjustable safety level. In *IEEE Conference on Emerging Technologies Factory Automation (ETFA 2009)*, 2009.

[35] Dawood Khan, Robert Davis, and Nicolas Navet. Schedulability Analysis of CAN with Non-abortable Transmission Requests. In *16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2011)*, Toulouse, France, September 2011. IEEE.

[36] Jan H. M. Korst, Emile H. L. Aarts, and Jan Karel Lenstra. Scheduling periodic tasks with slack. *INFORMS Journal on Computing*, 9(4) :351–362, 1997.

[37] S. Lauzac, R. Melhem, and D. Mossé. An improved rate-monotonic admission control and its applications. *IEEE Transactions on Computers*, 52(3) :337–350, 2003.

[38] José María López, José Luis Díaz, Joaquín Entrialgo, and Daniel García. Stochastic analysis of real-time systems under preemptive priority-driven scheduling. *Real-Time Syst.*, 40(2) :180–207, November 2008.

[39] J. Migge M. Boyer, N. Navet. An Efficient and Simple Class of Function to Model Arrival Curve of Packetised Flows. In *First International Workshop on Worst-Case Traversal Time (WCTT)*, 2011.

[40] M. Marouf, L. George, Y. Sorel, et al. Schedulability analysis for a combination of non-preemptive strict periodic tasks and preemptive sporadic tasks. *17th IEEE International Conference on Emerging Technologies and Factory Automation, ETFA'12*, 2012.

[41] S. Martin, P. Minet, and L. George. The trajectory approach for the end-to-end response times with non-preemptive fp/edf*. *Software Engineering Research and Applications*, pages 229–247, 2006.

[42] Patrick Meumeu Yomsi, Dominique Bertrand, Nicolas Navet, and Rob Davis. Controller Area Network (CAN) : Response Time Analysis with Offsets. In *9th IEEE International Workshop on Factory Communication System (WFCS 2012)*, 2012.

[43] J. Migge. *Timing Analysis of Real-Time Scheduling Policies : A Trajectory Based Model*. PhD thesis, November 1998.

[44] A. Mok and D. Chen. A multiframe model for real-time tasks. volume 23, pages 635–645, 1996.

[45] A. Monot, N. Navet, and B. Bavoux. Impact of clock drifts on CAN frame response time distributions. In *16th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA 2011)*, 2011.

[46] A Monot, N. Navet, B. Bavoux, and C. Maxim. Fine-grained Simulation in the Design of Automotive Communication Systems. In *Embedded Real Time Software and Systems (ERTSS'2012)*, February 2012.

[47] A Monot, N. Navet, B. Bavoux, and F. Simonot-Lion. Multicore scheduling in automotive ECUs. In *Embedded Real Time Software and Systems (ERTSS'2010)*, May 2010.

[48] Mouaaz Nahas, Michael J. Pont, and Michael Short. Reducing message-length variations in resource-constrained embedded systems implemented using the Controller Area Network (CAN) protocol. *J. Syst. Archit.*, 55 :344–354, May 2009.

[49] N. Navet, B. Delord, and M. Baumeister. Virtualization in automotive embedded systems : an outlook. Seminar at RTS Embedded Systems 2010 (RTS'2010), March 2010. Slides available at `http://www.realtimeatwork.com`.

[50] N. Navet, A. Monot, B. Bavoux, and F. Simonot-Lion. Multi-source and multicore automotive ECUs - OS protection mechanisms and scheduling. In *IEEE International Symposium on Industrial Electronics (ISIE)*, 2010.

[51] N. Navet, A. Monot, and J. Migge. Frame latency evaluation : when simulation and analysis alone are not enough. 8th IEEE International Workshop on Factory Communication Systems (WFCS2010), Industry Day, 2010.

[52] N. Navet and H. Perrault. Mécanismes de protection dans AUTOSAR OS. Seminar at RTS Embedded Systems 2009 (RTS'2009), April 2009. slides available at http://www.realtimeatwork.com/.

[53] N. Navet and H. S. Son. Performance and fault tolerance of real-time applications distributed over CAN (Controller Area Network). In *CiA-CAN in Automation*, 1997.

[54] N. Navet, Y.-Q. Song, and F. Simonot. Worst-case deadline failure probability in real-time applications distributed over controller area network. *J. Syst. Archit.*, 46(7) :607–617, April 2000.

[55] Nicolas Navet, Aurélien Monot, Bernard Bavoux, and Françoise Simonot-Lion. Multi-source and multicore automotive ECUs - OS protection mechanisms and scheduling. In *International Symposium on Industrial Electronics - ISIE 2010*, Bari, Italie, July 2010.

[56] Nicolas Navet and Ye-Qiong Song. Design of Reliable Real-Time Applications Distributed over CAN (Controller Area Network). In *INCOM98, IFAC 9th Symposium on Information Control in Manufacturing*, 1998.

[57] Thomas Nolte, Hans Hansson, and Christer Norström. Probabilistic worst-case response-time analysis for the Controller Area Network. In *the 9th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS '03)*, 2003.

[58] Y. Oh and H. S. Son. Tight performance bounds of heuristics for a real-time scheduling problem. Technical Report CS-93-24, Department of Computer Science, University of Virginia, 1993.

[59] Y. Oh and S.H. Son. Fixed-priority scheduling of periodic tasks on multiprocessor systems. Technical Report CS-95-16, Department of Computer Science, University of Virginia, 1995.

[60] OSEK Group. OSEK/VDX Operating System Specification . http://www.osek-vdx.org.

[61] RealTime-at-Work. NETCAR-Analyzer : a CAN analysis tool. disponible à http://www.realtimeatwork.com/software/netcar-analyzer/, 2009.

[62] RealTime-at-Work. NETCAR-ECU : a task scheduling configuration tool. disponible à http://www.realtimeatwork.com/, 2009.

[63] RealTime-at-Work. RTAW-Sim : a CAN simulation tool. disponible à http://www.realtimeatwork.com/software/rtaw-sim/, 2009.

[64] O. Redell and M. Törngren. Calculating exact worst case response times for static priority scheduled tasks with offsets and jitter. In *Real-Time and Embedded Technology and Applications Symposium (RTAS'02)*, 2002.

[65] M. Negrean K. Richter-M. Jersak-R. Ernst S. Schliecker, J. Rox. System Level Performance Analysis for Real-Time Automotive Multicore and Network Architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 979–992, 2009.

[66] Symtavision. SymTA/S : a Real-Time analysis tool. available at `https://symtavision.com/symtas.html`.

[67] K. Tindell, A. Burns, and A. J. Wellings. Calculating Controller Area Network (CAN) message response times. *Control Engineering Practice*, 3(8) :1163 – 1169, 1995.

[68] Haibo Zeng, Marco Di Natale, Paolo Giusto, and Alberto L. Sangiovanni-Vincentelli. Stochastic Analysis of CAN-Based Real-time Automotive Systems. *IEEE Trans. Industrial Informatics*, 5(4) :388–401, 2009.

[69] Haibo Zeng, Marco Di Natale, Paolo Giusto, and Alberto L. Sangiovanni-Vincentelli. Using statistical methods to compute the probability distribution of message response time in Controller Area Network. *IEEE Trans. Industrial Informatics*, 5(4) :678–691, 2010.

[70] A. Zuhily and A. Burns. Exact response time scheduling analysis of accumulatively monotonic multiframe real time tasks. pages 410–424, 2008.

[71] A. Zuhily and A. Burns. Exact scheduling analysis of non-accumulatively monotonic multiframe tasks. In *16th International Conference on Real-Time and Networked Systems (RTNS 2008)*, 2008.