THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut Supérieur de l'Aéronautique et de l'Espace (ISAE)

Présentée et soutenue par :
Cédric MAUCLAIR

le jeudi 13 juin 2013

Titre :

Une approche statistique des réseaux temps réel embarqués

École doctorale et discipline ou spécialité :
ED MITT : Réseaux, télécom, système et architecture

Unité de recherche :
Équipe d'accueil ISAE-ONERA MOIS

Directeur(s) de Thèse :

M. Bruno d'AUSBOURG (directeur de thèse)
M. Guy DURRIEU (co-directeur de thèse)

Jury :
M. Bruno d'AUSBOURG - Directeur de thèse
M. Guy DURRIEU - Co-directeur de thèse
M. Jean-Marie FOURNEAU - Rapporteur
M. Christian FRABOUL - Examinateur
M. Alain JEAN-MARIE - Examinateur
M. Nicolas NAVET - Examinateur
M. Ye-Qiong SONG - Rapporteur

Remerciements

Je tiens à remercier toutes les personnes que j'ai rencontrées au cours de mon doctorat et qui m'ont aidé à en voir le bout après cinq ans et demi de labeur : tout d'abord, mes directeurs Bruno et surtout Guy ; mes amis et les thésards que j'ai pu croiser ; ma famille bien–sûr et surtout mon `0:)`, *« je t'aime »*.

Soit, par ordre alphabétique, merci à Angélique, Bernard, Bruno, Claire, Didier, Dimitri, Florian, Géraldine, Guillaume, Guy, Jean-Loup, Julien, Laurie, Ludovic, maman, Maria, Michael, Nathalie, Pierre, Stéphanie, Sylvain, Thomas et tous ceux que j'ai oublié...

Table des matières

Annexes

Liste des figures

Liste des tableaux

Liste des abréviations

ABS	*Anti-Blocking System*
AFDX	*Avionics Full-DupleX switched Ethernet*
ARINC	*Aeronautical Radio, Incorporated*
ARP–4754	*Certification Considerations for Highly-Integrated Or Complex Aircraft Systems*
BAG	*Bandwidth Allocation Gap*
BDD	*Binary Decision Diagram*
CAN	*Controler Area Network*
CERN	*European Organization for Nuclear Research*
DAL	*Design Assurance Level*
DITS	*Mark 33 Digital Information Transfer System*
DiffServ	*Differentiated Service*
DME	*Distance Measuring Equipment*
DO–178	*Software Considerations in Airborne Systems and Equipment Certification*
DO–254	*Design Assurance Guidance For Airborne Electronic Hardware*
DO–297	*Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations*
DP	*Dynamic Priority*
FIFO	*First In First Out*
FP	*Fixed Priority*
GPS	*Global Positioning System*
HPF	*Highest Priority First*
IDAL	*Item Design Assurance Level*
IEEE	*Institute for Electrical and Electronics Engineers*
IMA	Avionique Modulaire Intégrée
IP	*Internet Protocol*
MPLS	*MultiProtocol Label Switching*
NTP	*Network Time Protocol*
PRISM	*PRISM*
PTP	*Precision Time Protocol*

SMV	*Symbolic Model Verification*
SSM	*Sign Status Matrix*
UDP	*User Datagram Protocol*
UPPAAL	*UPPsala Universitet & AALborg University*
VL	*Virtual Link*
VOR	*VHF Omnidirectional Range*
WCET	*Worst Case Execution Time*
WCRT	*Worst Case Response Time*
WCTT	*Worst Case Transmit Time*

Introduction I

Depuis l'arrivée des réseaux de nouvelle génération, des réseaux de communication toujours plus vastes et plus complexes sont déployés au cœur des systèmes avioniques embarqués à bord d'aéronefs (avions, hélicoptères, satellites, etc.). Ces systèmes ont aujourd'hui la charge de fournir un nombre croissant de fonctionnalités et de services aux membres d'équipage afin de faciliter les différentes phases de vol : décollage, croisière et atterrissage, vols stationaires (missions de recherche ou de secours médical par exemple). Sans compter les missions militaires.

À titre d'illustration, le nombre d'équipements qui composent le système avionique d'un avion Airbus A310 (premier vol en 1983) est de 77 contre 115 pour l'A340 dont le premier vol a eu lieu huit ans plus tard. Dans le même temps, le nombre de bus numériques a triplé passant de 136 à 368. Dans les années 2000–2010, la part de logiciel embarqué (dont le plus gros contributeur est le système avionique) est passée de 20Mo sur l'A340 à 100Mo sur l'A380. Quant à la puissance de calcul, elle a été multipliée par quatre.

La complexité de ces systèmes croit très vite à mesure que de nouveaux équipements s'y voient connectés pour apporter de nouvelles fonctionnalités, en améliorer, en compléter et/ou en remplacer d'autres. Aujourd'hui par exemple, les GPS (*Global Positioning System*) sont de plus en plus utilisés pour la navigation, en plus des systèmes traditionnels que sont le VOR pour le positionnement azimutal autour d'une balise (*VHF Omnidirectional Range*) et le DME pour l'éloignement par rapport à cette même balise (*Distance Measuring Equipment*). Si les bus numériques mis au point dans les années 1970–1980 avaient dû être utilisés, l'Airbus A380 n'aurait peut–être jamais vu le jour. En effet, le câblage aurait été bien trop important et trop lourd. Ce dernier, dans sa version comprenant les réseaux numériques de nouvelles génération réduisant le nombre et la masse, représente déjà près de 530km contre 300km pour un A340.

Le avionneurs ont cherché des alternatives et se sont tournés vers le multiplexage des informations à haut débit sur les mêmes câbles physiques afin d'en réduire le nombre et le poids total. Ces bus numériques de nouvelle génération, et en particulier les réseaux AFDX (*Avionics Full-DupleX switched Ethernet*), sont des réseaux où se croisent les informations entre les différents éléments qui y sont connectés. Ce multiplexage introduit des retards, en

partie compensés par des débits largement supérieurs aux technologies précédentes, d'un facteur 10 à 1000, et par une flexibilité accrue. Néanmoins, et malgré de nombreux avantages, des délais persistent et dépendent du trafic : plus il est important, plus le délai de traversée du réseau est élevé. Nous présentons ces problématiques au chapitre II.

Par ailleurs, les avionneurs sont légalement tenus de borner tous les délais introduits par des communications au sein de leurs aéronefs. Ce qui n'était alors pas un problème avec les anciens bus numériques est devenu un sujet de recherche fertile depuis quelques années : l'étude et la preuve du déterminisme des réseaux embarqués.

À l'heure où ces lignes sont écrites et à la connaissance de l'auteur, seules deux méthodes formelles sont considérées comme des moyens acceptables de calculer et prouver des bornes temporelles sur les réseaux AFDX : le calcul réseau et la méthode des trajectoires (ou holistique). Ces théories mathématiques permettent de prouver que les bornes calculées sont bien des bornes absolues qui ne seront jamais dépassées en condition de fonctionnement nominal. C'est un résultat fondamental qui a permi l'utilisation de réseaux multiplexés de type Ethernet au sein d'environnements critiques tel que des avions ou des hélicoptères. La théorie du calcul réseau est dûe à Cruz [22, 23] et la première application aux réseaux AFDX est dûe à Jérôme Grieu [30]. Depuis, de nombreuses études sont venus compléter et améliorer ces premiers résultats. Au chapitre III, nous présentons les théories du calcul réseau et de la méthode des trajectoires. Nous y dressons également un panorama le plus large possibles des méthodes utilisées pour l'analyse de propriétés temporelles des réseaux AFDX.

Malgré une certitude mathématique que ces bornes ne seront jamais dépassées en fonctionnement nominal, elles sont souvent bien plus larges que les valeurs maximales observées lors de longues simulations : de l'ordre de dix fois supérieure en fait. La raison en est simple. Les hypothèses faites par ces théories, et qui permettent de mener des calculs analytiques, sont trop simplificatrices par rapport à la complexité des phénomènes mis en jeu. De fait, elles conduisent à des résultats dits *pessimistes*. De plus, ces simulations révèlent aussi que les conditions conduisant aux scénarios les plus néfastes à une transmission rapide des messages sur un réseau sont très rares. Autrement dit, ces théories permettent de caractériser une surestimation parfois très large du cas pire : c'est-à-dire conduisant aux délais maximaux. Cela ne donne qu'une très faible visibilité sur « les performances » d'un réseau.

Nous sommes convaincus qu'une bonne conception et une bonne configuration passent par une meilleure compréhension de la dynamique d'un réseau : du meilleur au pire cas. C'est pourquoi ces travaux consistent en la mise en place d'une méthode précise d'analyse de propriétés temporelles de réseaux AFDX basée sur des méthodes statistiques de type Monte Carlo. Pour cela, nous procédons en trois étapes.

> **Remarque** Tout d'abord, nous faisons dans ce manuscrit un parallèle très fort entre un système temps réel et un réseau (en tant qu'ensemble de systèmes temps réel interconnectés). Ces notions seront utilisées indifféremment dans le texte, tout en respectant le contexte. Cet élément important est rappelé au chapitre IV.

À la première étape, présentée au chapitre V, nous proposons une méthode de calcul de la loi de distribution des délais subis par une instance d'une tâche temps réel parmi n. Pour cela, nous mettons en évidence un phénomène d'explosion combinatoire de la complexité du problème en $\mathcal{O}(n \cdot n!)$. Puis, nous introduisons la notion de « motifs » afin de représenter les données d'une manière plus compacte et d'abaisser la complexité à $\mathcal{O}(n \cdot 2^n)$. Nous abaissons encore cette complexité en tirant parti de certains caractères communs entre les tâches et les instances du système au travers de classes d'équivalence. Elle varie alors entre $\mathcal{O}(n^2)$ et $\mathcal{O}(n \cdot 2^n)$.

Dans un deuxième temps, au chapitre VI, nous proposons un algorithme efficace de calcul de la loi de distribution des délais subis par une tâche strictement périodique au sein d'un système temps réel dont les déphasages initiaux sont connus. Cet algorithme utilise la notion de trace symbolique des activations des tâches. Dans un premier temps, nous utilisons cette trace symbolique pour calculer certaines propriétés temporelles du système : l'arriéré de travail et les délais minimaux et maximaux des tâches. Puis, nous présentons notre algorithme original en exploitant directement les résultats de la phase précédente.

Enfin, au chapitre VII, nous évaluons la loi de distribution des délais subis par des tâches temps réel à l'aide d'une méthode statistique de type Monte Carlo. Pour cela, nous montrons que le nombre de déphasages initiaux entre les tâches interdit toute résolution directe du problème. Puis, après avoir rappelé les grands principes des méthodes de Monte Carlo, nous proposons un mode opératoire flexible. Autour d'un exemple, nous discutons de la pertinence de nos résultats en les comparant aux délais pire cas calculés pour les tâches du système par les méthodes formelles classiques (présentées au chapitre III).

1

Contexte, modèle et notations

Réseaux avioniques et systèmes temps réel

II

À l'instar de ce qui se passe dans d'autres secteurs de l'économie, la réduction des coûts de développement, la modularité, la réutilisabilité ou encore le partage des ressources sont des enjeux majeurs pour l'industrie aéronautique. De plus, la taille et la complexité croissantes des avions de nouvelle génération — Airbus A380, A400M, futur A350 et Boeing 787 *Dreamliner* —, imposent d'avoir recours à des réseaux de communication de données toujours plus vastes, plus souples et plus performants afin de supporter des trafics en constante progession.

Les coûts de développement et de maintenance d'un réseau avionique entièrement basé sur le bus ARINC 429 auraient été beaucoup trop élevés pour un avion comme l'A380. Cela sans compter le surpoids engendré par les nombreux câbles nécessaires pour relier les différents équipements de l'appareil, ni les problèmes liés à la redondance de ces câbles dans un espace aussi restreint qu'un avion et à plus forte raison un hélicoptère. Le bus ARINC 429 est détaillé à la section 2.2.

Au début des années 2000, les industriels du secteur aéronautique ont standardisé et transposé à l'aviation civile le concept d'Avionique Modulaire Intégrée (IMA). L'AFDX (*Avionics Full-DupleX switched Ethernet*) est un moyen de communication central qui participe aux architectures avioniques en offrant des garanties temporelles aux flux de ses abonnés. Il est formellement décrit dans la partie 7 du standard aéronautique ARINC 664. La section 2.3 détails ces concepts d'IMA et de réseaux AFDX. Cette section ne prétend pas à l'exhaustivité et ne présente que les caractéristiques essentielles utiles aux idées développées dans ce manuscrit.

2.1 Qu'est-ce qu'un réseau avionique ?

Les événements redoutés au niveau d'un appareil (avion, hélicoptère, drône, satellite, etc.) sont classés par sévérité. Il existe cinq paliers, définis dans le guide de développement des systèmes complexes ARP-4754 [3] :

- *catastrophique* : entraîne plusieurs défaillances majeures et/ou la perte complète de l'appareil et/ou de membres de l'équipage ;
- *dangereux* : sévère impact négatif sur la sécurité du vol, réduit la capacité de l'équipage à contrôler les conditions de vol en raison de problèmes d'ordre physique ou de surcharge de travail, entraîne des blessures graves et/ou fatales parmi les passagers de l'appareil ;
- *majeur* : réduit les marges de sécurité et/ou les fonctions du système et/ou les conditions de pilotage, engendre une surcharge de travail gérable pour l'équipage, entraîne des blessures parmis les occupants de l'appareil, détérioration physique de l'habitacle ;
- *mineur* : la panne est ressentie mais n'a pas d'effet sur la sécurité du vol, tout au plus, les passagers ressentent un léger inconfort ;
- *sans effets* : aucun impact sur la sécurité du vol.

Après l'identification de l'ensemble des événements redoutés au niveau de l'appareil, un degré de criticité est affecté aux différentes fonctions de l'appareil suivant l'impact au niveau appareil d'une défaillance où d'un fonctionnement erroné non-détecté de ces fonctions. Pour mitiger ces risques, les autorités imposent des objectifs quantitatifs, sous la forme de probabilités maximales d'occurence de panne ou de comportements erronés par heure de vol (10^{-9}, 10^{-7}, etc.), et qualitatifs sous la forme de règles de développement : les DAL, pour *Design Assurance Level*.

Ensuite, des DAL de niveau équipements (appelés IDAL) sont redescendus selon les choix d'architecture pour réaliser les différentes fonctions. Les règles de développement pour les composants avioniques en fonction des DAL sont décrites dans plusieurs documents. Citons le DO-178 [4] qui fixe les règles de développement pour les composants logiciels, le DO-254 [1] qui fixe celles des composants électroniques (et matériels en général) et le DO-297 [2] qui décrit les règles relatives au développement de systèmes IMA ; il y en a beaucoup d'autres.

Les réseaux avioniques sont avant tout chargés d'acheminer des données entre différents équipements d'un aéronef et se retrouvent au cœur de nombreuses fonctions de différentes criticités. De fait, ils héritent de la criticité la plus élevée parmi les fonctions qu'ils interconnectent. Généralement, la perte totale des communications conduit à un événement catastrophique. C'est la raison pour laquelle les différents constituants d'un réseaux AFDX sont développés aux niveaux de DAL les plus élevés (A), pour les composants logiciels et matériels.

Bien que développés selon des règles très stictes, les autorités imposent des solutions de repli, ou *backup* en anglais, lorsque la fonction est de criticité maximale. Pour cette raison notamment, un autre bus numérique est toujours utilisé en solution de secours d'un réseau AFDX : le bus de communication ARINC 429 que nous présentons à la section suivante. Ce n'est pas la seule raison de garder ce bus. En effet, c'est aujourd'hui le plus utilisé dans l'aéronautique et la transition prendra du temps pour être achevée.

En plus des contraintes de développement, les autorités de certification fixent également (et surtout) des exigences opérationnelles aux équipements. Les travaux présentés dans ce manuscrit ne traitent que de ce type d'exigences. Plus particulièrement, l'une d'elles est de connaître le temps maximal mis par toute donnée pour traverser le réseau. Un message

transporte généralement plusieurs données élémentaires d'un applicatif à un autre et peut être décomposé en plusieurs trames s'il est trop long. La trame est l'unité de transport élémentaire au niveau réseau et sa taille est bornée par le protocole utilisé. La détermination des temps de traversée maximal de bout en bout permet notamment de détecter certaines défaillances et/ou comportements erronés du réseau et/ou des équipements eux-mêmes.

2.2 Bus de communication ARINC 429

Le standard aéronautique ARINC 429, publiée pour la première fois en 1977, décrit un bus de communication de données connu sous le nom de DITS (*Mark 33 Digital Information Transfer System*) ; par commodité et abus de langage, on parle du bus ARINC 429. Il équipe tous les types d'avions et d'hélicoptères. Dans le domaine des systèmes avioniques civils et jusqu'à l'arrivée du gros porteur A380 et du B787, il était le bus privilégié pour assurer les communications entre équipements à bord des aéronefs.

Le bus ARINC 429 est de type point-à-multipoints, monoémetteur et multirecepteurs (*multicast*) : un seul émetteur et jusqu'à vingt récepteurs. Les données, ou *mots*, sont composées de 32 bits. Chaque mot s'interprète de la façon suivante (voir la figure 2.1, extraite de [61]) :

- bits 1 à 8 : un label pour identifier le type de la donnée ;
- bits 9 à 10 : deux bits pour identifier la source et la destination de la donnée ;
- bits 11 à 29 : dix-neuf bits de donnée utile ;
- bits 30 à 31 : deux bits appelés SSM (*Sign Status Matrix*) qui permettent de contrôler l'état de la donnée : normale, invalide, non-calculée et erreur détectée (pour les détails, voir *The Avionics Handbook* [61, chapitre 2]).

Il existe deux débits : un débit à basse vitesse entre 12 et 14.5 kilobits par seconde et un débit à « grande vitesse » à 100 kb/s.

Les principaux avantages de l'ARINC 429 sont sa simplicité — de part sa topologie point-à-multipoints et l'organisation de ses trames — et sa robustesse — dûe en grande partie à cette simplicité. Ses principaux inconvénients sont une faible charge utile (dix-neuf pour trente-deux) et un faible débit. Par ailleurs, la multitude de câbles rend difficile tout déplacement d'un équipement et alourdit les appareils.

Lors de sa mise en service, ces inconvénients étaient mineurs, voire inexistants compte tenu des volumes d'informations échangés. Aujourd'hui, sur les appareils de nouvelle génération où les volumes sont beaucoup plus importants, ils posent de réels problèmes de conception et de maintenance. En effet, compte tenu de la nature *multicast* du bus ARINC 429 et du fait que chaque équipement envoie ses propres données, on peut estimer le nombre de câbles nécessaire à un réseau complet d'A320 ou d'A330/A340 aux alentours de 200, ce nombre passe à plus de 1000 pour un avion comme l'A380. Le surpoids engendré aurait été

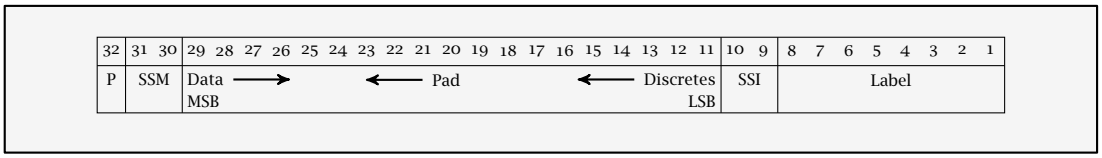

Figure 2.1 Mot de données ARINC 429 (en bits).

trop important. Par ailleurs, la modularité du bus ARINC 429 étant quasi–nulle, la problématique de trouver des chemins de passage redondés, ségrégués et sûrs devenait un défi praticulièrement complexe à relever.

Cependant, et malgré ses inconvénients, l'ARINC 429 est encore présent à bord de l'A380 où il est utilisé comme réseau de secours du réseau AFDX. En effet, une panne ou un comportement erroné simple (le réseau AFDX) ne doit pas conduire à un événement catastrophique. La perte des communications étant critiques, il faut deux technologies disymétriques pour tenir l'exigence. En plus de cela, le protocole AFDX venait tout juste de voir le jour et les autorités de certification avaient besoin d'être rassurées.

2.3 Avionique modulaire intégrée et réseaux AFDX

L'IMA est un concept apparu dans les années 1990 chez les avionneurs militaires, puis développé et standardisé dans les années 2000 par les industriels du secteur aéronautique afin de modulariser la conception et le développement de leurs avions. L'IMA est censée répondre à deux problématiques majeures : (1) la réduction des coûts de conception, de développement et de maintenance des différents composants des systèmes avioniques et (2) l'augmentation de leur modularité et réutilisabilité. Pour cela, l'accent est mis sur le développement de calculateurs génériques, constitués de composants standards et capables d'héberger simultanément toutes sortes d'applications différentes afin de pouvoir réutiliser ces calculateurs dans différents modèles d'appareils, sans nouveaux coûts de développement (ou peu).

Un effort est également porté sur le réseau de communications afin d'alléger les appareils et d'accroître la modularité des calculateurs, en leur permettant d'être déplacés plus facilement dans l'appareil, lors d'une nouvelle conception par exemple. Ce nouveau réseau est décrit par le standard aéronautique ARINC 664 ; l'AFDX n'en est qu'une implémentation particulière, décrite dans la partie 7 du standard. La figure 2.2 illustre la structure d'une trame (à mettre en parallèle de la figure 2.1, attention au facteur huit pour passer des octets aux bits). L'AFDX apporte une solution aux trois inconvénients majeurs de l'ARINC 429 :

1. la charge utile passe de 19 bits à environ 1500 octets (1 octet est égal à 8 bits) ;
2. le débit passe de 100 kb/s à 100 Mb/s (on évoque même le Gb/s aujourd'hui) ;
3. enfin, le nombre de câbles utilisés pour relier les différents calculateurs est grandement réduit grâce à une architecture en étoile et au partage des ressources par l'utilisation de techniques de multiplexage.

2.3.1 Architecture physique

La technologie retenue pour l'AFDX repose sur les protocoles *Ethernet full-duplex* com-

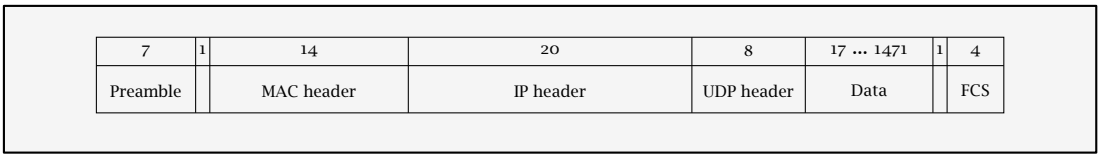

Figure 2.2 Structure d'une trame AFDX (en octets).

muté, IP et UDP, et sur l'utilisation de contrats de trafic afin d'assurer une qualité de service aux flux du réseau, appelés *liens virtuels* ou VL. Le protocole *Ethernet* est défini par l'*Institute for Electrical and Electronics Engineers* dans le standard IEEE 802.3 ; il est largement répandu et permet des coûts de développement relativement faibles. Ce protocole est utilisé partout aujourd'hui : entreprises comme particuliers. Les débits théoriques vont jusqu'à 100 Mb/s pour la solution retenue ; il est envisagé 1 Gb/s pour une prochaine évolution. L'utilisation de commutateurs et de câbles *full-duplex* permet de supprimer les collisions de trames inhérentes à la technologie *Ethernet*. Ainsi, l'indéterminisme lié à la réémissions de trames suite à une collision n'existent plus. Néanmoins, il reste l'indéterminisme lié à la gestion des trames au niveau des commutateurs (voir plus loin).

Le protocole *User Datagram Protocol* permet une gestion des envois et des réceptions des données par les différentes applications de manière ségréguée, par l'intermédiaire de ports de communication qui peuvent être partagés en lecture ou en écriture sous certaines conditions. Le protocole *Internet Protocol* permet quant à lui de procéder à l'échange de messages plus grands que la taille maximale autorisée par le contrat de trafic d'un VL. Les messages sont fragmentés en plusieurs trames qui seront réassemblées à l'arrivée, puis transmis aux applications via les ports UDP.

Enfin, comme le montre la suite, si les architectures matérielles sont différentes, d'un point de vue purement logique, l'ARINC 429 et l'AFDX sont très proches (la taille des données mise à part). Cela garantit une transition plus souple d'une technologie à l'autre, notamment pour les applications qui peuvent continuer à fonctionner sur les mêmes principes qu'auparavant.

Un réseau AFDX est constitué d'équipements reliés entre eux par des commutateurs en une structure plus ou moins étoilée (voir la figure 2.3). Ces équipements sont connectés au réseau par le biais des *end-system*. On dit qu'un réseau AFDX est localement synchrone (au niveau des équipements) et globalement asynchrone (entre les équipement et les commutateurs). Ces équipements accèdent au réseau via leurs interfaces nommées *end-system* et dont le rôle est de gérer l'accès au réseau, de réguler le trafic en sortie de l'équipement en accord avec les contrats de trafic établis statiquement et enfin de récupérer les données pour les applications de son équipement. Les *end-systems* sont connectés à deux commutateurs duaux (qui sont les symétriques l'un de l'autre) et constituent les nœuds d'entrée du réseau.

Les nœuds intérieurs du réseau sont les commutateurs et ils ont la charge d'assurer le routage statique (défini à l'avance) des trames AFDX au sein du réseau. Ils sont constitués d'un ensemble de ports d'entrée/sortie qui les relient à des *end-systems* ou d'autres commutateurs et d'une mémoire (aussi appelée *buffer*). Chaque port est composé de deux files d'attente de priorités différentes, destinées à stocker des pointeurs vers les trames stockées en mémoire. Dans certaines implémentations, *une « tête de lecture » scanne régulièrement chaque port d'entrée en séquence* ; lorsqu'une trame arrive dans l'un des ports d'entrée, elle est écrite en mémoire et son adresse est stockée dans les ports de sortie et les files d'attente correspondant à ses destinations et sa priorité respectivement. Le fait qu'une trame arrivée puisse ne pas être prise en compte immédiatement par le commutateur en raison d'une lecture séquentielle est le point de départ de ces travaux (voir le chapitre VII).

Les câbles utilisés sont *full-duplex* et composés de deux paires torsadées blindées, ce qui garantit un signal clair et une bonne immunité aux perturbations électromagnétiques.

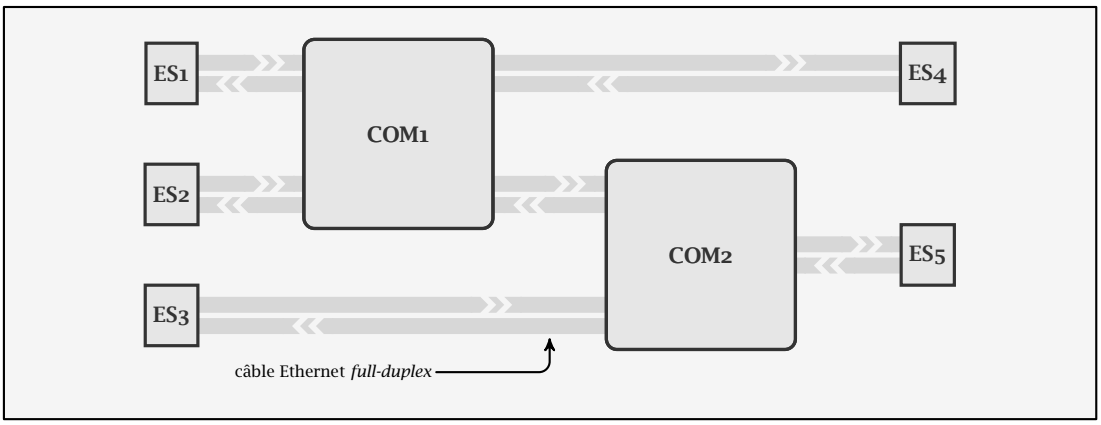

Figure 2.3 Vue physique d'un réseau AFDX.

Ils offrent également une certaine souplesse dans le choix du placement des équipements à l'intérieur des aéronefs, contrairement au bus ARINC 429 qui nécessite de nombreux câbles qu'il faut tirer dans tout l'appareil. Bien-sûr, l'utilisation des commutateurs alourdit un appareil, mais les gains de câblage compensent largement. En pratique, il existe des passerelles vers différents bus, ARINC 429 et ARINC 825 (CAN) pour n'en citer que deux.

La figure 2.3 est une représentation d'un réseau AFDX simple : quatre équipements représentés par leurs *end-systems*, deux commutateurs, et six câbles *full-duplex*, représentés par deux traits épais entre l'équipement et son commutateur avec un chevron pour indiquer le sens de circulation sur le câble.

2.3.2 Architecture logique

Afin d'assurer la compatibilité et la transition du bus ARINC 429 vers les réseaux AFDX, le niveau logique de ces derniers mime le comportement du premier. Une utilisation conjointe des deux technologies simultanément est possible grâce à des passerelles ; c'est le cas à bord de l'A380 par exemple. D'autre part, ce niveau logique permet d'abstraire complètement la couche réseau des applications qui n'ont pas connaissance du *medium* utilisé. (En théorie du moins. Dans la pratique, cela suppose l'utilisation d'un *middleware* générique entre les couches réseau et applicatives ou l'utilisation d'un standard de communication. Cependant, ce n'est pas l'objet de ces travaux.)

Cette abstraction du réseau physique est réalisée à l'aide de liens virtuels *multicast* appelés VL (pour *Virtual Link* en anglais). Il s'agit de canaux de communication qui relient une source émettrice à plusieurs récepteurs. Plusieurs données issues d'une même source peuvent être acheminées par un même VL (cela est même recommandé pour assurer une bonne utilisation de ces derniers). Contrairement à l'ARINC 429, les récepteurs ne sont pas limités en nombre. Les « chemins » empruntés par les VL sont définis comme un équipement source — la même que le VL —, une suite de commutateurs, et un équipement faisant partie des destinations du VL. Un VL peut avoir plusieurs chemins, mais toujours sous la forme d'un arbre (voir les VL 1 et 3 sur la figure 2.4).

Le rôle d'un *end-system* est de réguler le trafic des VL en fonction de leurs contrats de trafic. Ces contrats définissent la taille maximale des trames, $S_{\max}$, que peut envoyer

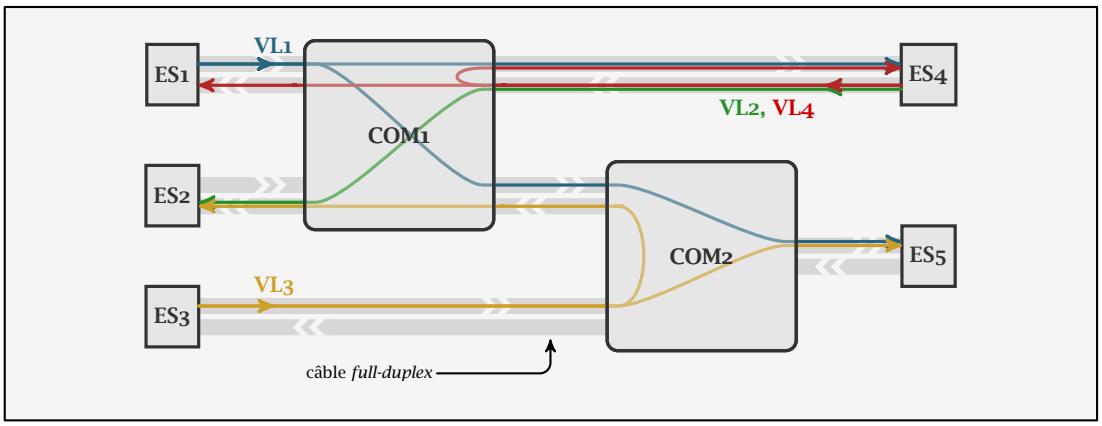

Figure 2.4 Vue logique d'un réseau AFDX.

un VL. Ils définissent également l'intervalle de temps minimal qui doit séparer deux trames consécutives, il est nommé BAG (*Bandwidth Allocation Gap*). De cette manière, même si une application « s'emballe » et se met à envoyer ses données trop rapidement, le *end-system* régule son trafic afin de ne pas saturer le reste du réseau et risquer de léser les autres utilisateurs du réseau, les abonnés ou VL.

Ensuite, les commutateurs sont chargés du routages des trames vers leurs destinations en fonction du VL qui les transporte. À leur(s) arrivée(s), les trames sont placées dans des files d'attente dans les ports qui les enverront vers les commutateurs suivants ou leurs équipements de destination. Il y a deux files d'attente qui représentent les deux niveaux de priorités que peuvent avoir les VL : *H* (priorité haute) et *L* (priorité basse).

La figure 2.4 reprend l'exemple de réseau AFDX simple vue à la figure 2.3 et y ajoute quatre VL. Le VL 1 a pour source le *end-system* 1 et pour destinations les *end-systems* 4 et 5. Pour cela, son chemin transite respectivement par les commutateurs 1 puis 2. Le VL 2 suit le chemin *end-system* 4, commutateur 1 et *end-system* 2 ; le VL 3 suit les chemins *end-system* 3, commutateurs 2 et 1 et *end-system* 2 et *end-system* 3, commutateur 2 et *end-system* 5 ; enfin le VL 4 suit les chemins *end-system* 4, commutateur 1 et *end-system* 1 et 4 de nouveau.

Sur le tronçon qui relie le commutateur 2 au *end-system* 5, les VL 1 et 3 sont multiplexés. Le multiplexage consiste à combiner les flux de données de telle sorte que plusieurs VL utilisent le même câble physique. Concrètement, il est réalisé en envoyant les trames lorsqu'aucune autre trame n'est en train d'être envoyée, sinon, elles sont mise en attente (chacun son tour). Sur le tronçon qui relie le commutateur 1 au *end-system* 4, les trames des VL 1 et 4 d'une part et 2 et 4 d'autre part se croisent sans collisions grâce à l'utilisation des câbles *full-duplex*.

Le multiplexage des VL dans un réseau AFDX et des débits plus importants permettent de réduire grandement le nombre de câbles nécessaires par comparaison à un réseau ARINC 429 équivalent. De fait, l'AFDX contribue à réduire le poids des avions. Mais le multiplexage est également le principal responsable de l'indéterminisme de ces réseaux. En effet, puisque les trames sont parfois mises en attente, il est difficile de prévoir le temps que met une trame particulière pour aller d'un équipement à un autre.

2.3.3 Une solution à tous les maux ?

Les réseaux AFDX apportent une solution partielle aux différents problèmes rencontrés par les industriels de l'aéronautique dans les années 2000 :

- ils réutilisent certains composants standards du commerce (*Ethernet full-duplex*, câbles, contacts et connecteurs standards dans l'aéronautique) et ont ainsi des coups de développement et de maintenance réduits par rapport à plusieurs solutions antérieures ;
- ils permettent d'alléger les avions et par là même leur consommation de carburant ;
- et ils permettent l'échange rapide d'une grande quantité de données.

Malgré ces avantages, les réseaux AFDX nécessitent toujours une solution de secours pour tenir les exigences fixées par la criticité des fonctions qui en dépendent. S'ils ont déjà apportés un gain de poids significatif, ce dernier pourrait être plus important encore si d'autres technologies semblables étaient utilisées pour remplacer définitivement le bus ARINC 429.

D'autre part, les coûts de développement et de certification des commutateurs sont particulièrement élevés. De plus, malgré l'utilisation de câbles *full-duplex* et, de fait, la suppression des collisions de trames, le multiplexage et le partage des moyens de communications induisent de l'indéterminisme sur les temps de traversée de bout en bout — les délais subis par les trames d'un équipement à un autre en traversant le réseau. Cet indéterminisme s'entend au sens où ils ne sont pas constants ; en effet, chaque trame d'un VL peut traverser le réseau en plus ou moins de temps, en fonction du trafic.

Cela dit, il est tout de même possible de borner ces délais de bout en bout des trames d'un VL et donc d'utiliser ces réseaux dans un environnement embarqué critique. Il existe différentes méthodes : principalement le calcul réseau, présenté pour la première fois par Cruz en 1991 [22, 23, 38] et la méthode des trajectoires, introduite par Martin en 2004 [9, 48]. Le chapitre III présente ces méthodes et comment elles sont appliquées pour le calcul des bornes des délais de bout en bout dans un réseau AFDX.

2.4 Qu'est-ce qu'un système temps réel ?

Un *système* est communément défini comme un ensemble d'agents dont les interactions ont pour objectif de conduire à la réalisation d'un but commun. Dans ces travaux, nous définissons un *système* comme un ensemble *de ressources* et *de fonctions* dont les bonnes exécutions conduisent à la réalisation d'un but commun. Les ressources y sont indivisibles, et il faut les partager « dans le temps » entre les différentes fonctions. En aéronautique par exemple, les buts peuvent être de piloter un avion, de calculer les paramètres de vol ou d'assurer les communications entre les différents équipements de l'appareil. Les ressources associées sont par exemple l'équipage, les équipements et le réseau avionique. Respectivement, les fonctions sont par exemple de contrôler les instruments, de calculer la vitesse de l'avion et de transmettre cette vitesse sur les instruments, à l'attention des pilotes.

Le terme *temps réel* est apparu avec les premières simulations. À l'époque, on parle de simulations *temps réel* lorsque celles-ci évoluent « en même temps » que les phénomènes simulés (le temps simulé est alors très proche du temps réel qui s'écoule). Aujourd'hui, la notion de *temps réel* est plutôt associée à celle de réactivité. En informatique industrielle

par exemple, les systèmes temps réel, ou cyber–physiques, sont utilisés pour contrôler ou piloter des procédés physiques soumis à des contraintes temporelles : pilote automatique, système de freinage ABS, appareil de régulation de température, etc. Du fait de cette relation, un système informatique temps réel est appelé à répondre à des sollicitations simultanées mais d'importance variable, ce qui se traduit de façon interne par un ensemble de tâches en concurrence et de priorités différentes. Produire des résultats exacts *et le faire dans des délais imposés* est ce qui caractérise les systèmes temps réel.

De nos jours, les systèmes temps réel sont omniprésents dans l'industrie. Citons le secteur aéronautique au travers des systèmes de navigation ; le secteur nucléaire via le contrôle des cœurs des réacteurs ; l'automobile et ses systèmes d'aide à la conduite ; les salles de marché où le traitement des données boursières se fait en « temps réel » et les *media* pour la diffusion de contenus numériques à la demande… La liste est loin d'être exhaustive.

En informatique, lorsque l'on parle de systèmes temps réel, on utilise le terme d'ordonnançabilité plutôt que de correction temporelle. Ainsi, un système temps réel est *ordonnançable* s'il respecte toutes ses échéances temporelles ; c'est–à–dire si toutes les tâches dont il a la charge sont exécutées en temps et en heure, ou encore si le temps de réponse de chacune de ses tâches (durée qui sépare l'activation de la fin effective d'exécution de la même instance) est inférieur à son échéance. Ainsi, une question fondamentale pour ces systèmes est la suivante : « Comment prouver qu'un système est ordonnançable et respecte toutes ses échéances ? »

> **Remarque** Dans la suite de ce manuscrit, les résultats sont supposés conformes aux spécifications fonctionnelles du système (les calculs sont bons) et seul l'aspect temporel de la correction d'un système est considéré. Autrement dit, ces travaux ne s'intéressent qu'à l'étude de l'ordonnançabilité des systèmes temps réel.

2.5 Modélisation d'un système temps réel

Une tâche est une unité de traitement d'un système temps réel : récupérer la mesure de température, calculer la commande, actionner les verrins, signaler l'alerte incendie, etc. Une instance est une activation particulière d'une tâche : la récupération de la température à l'instant t, le calcul de la commande demandée à l'instant t', etc.

Classiquement, les tâches et leurs instances sont décrites par un ensemble de propriétés (voir la figure 2.5 pour une illustration). Toutes les notions sont précisées ici ou au chapitre IV. On note τ_i la tâche i et ses propriétés sont :

- une période T_i : quantité qui caractérise la durée qui sépare l'activation de deux instances consécutives de la tâche. Cette durée peut être constante pour les tâches périodiques, représenter un minimum pour les tâches sporadiques ou être non définie pour les tâches apériodiques ;
- un pire temps d'exécution C_i ou temps d'exécution pire cas : quantité qui borne le temps maximal nécessaire pour exécuter n'importe quelle instance de la tâche, quelque

soient les conditions d'exécution. Le calcul de ce pire temps d'exécution, couramment appelé WCET, acronyme anglais de *Worst Case Execution Time*, fait l'objet de nombreuses études ;

- une priorité P_i : détermine l'ordre dans lequel les tâches seront exécutées par le système ; plus le terme P_i est petit, plus la priorité de la tâche i est élevée ;
- une échéance relative D_i : intervalle de temps après une activation de la tâche pendant lequel l'instance activée doit être exécutée pour être valide. On dit alors qu'elle respecte son échéance. Si ce n'est pas le cas, le système n'est pas ordonnançable ;
- un pire temps de réponse R_i ou temps de réponse pire cas : quantité *calculée* qui correspond à la plus longue durée qui sépare l'activation de la fin d'exécution effective de l'une des instances de la tâche. C'est une borne supérieure, souvent appelée WCRT, acronyme anglais de *Worst Case Response Time*. Pour être ordonnançable, R_i doit être inférieur à D_i pour toutes les tâches.

On note $J_{i,j}$ l'instance j de la tâche i et ses propriétés sont :

- une date d'activation $r_{i,j}$ dans le système : correspond à la date à laquelle l'instance requiert les ressources pour son exécution ;
- une date de début d'exécution $w_{i,j}$: correspond au premier instant où l'instance est exécutée pour la première fois. S'il existe une instance plus prioritaire dans le système à l'activation de l'instance $J_{i,j}$, alors les dates de début d'exécution et d'activation sont différentes ;
- une échéance absolue $d_{i,j}$: définie directement à partir de la date d'activation et de l'échéance relative de la tâche correspondant à l'instance : $d_{i,j}$ est égal à la somme de $r_{i,j}$ et D_i.

Toutes les tâches du système partagent une même ressource : le processeur. Ce dernier est alloué aux instances par le système selon une politique de gestion des priorités « plus forte priorité d'abord ». Les ressources peuvent être allouées (1) pour toute la durée d'exécution des instances (c'est-à-dire C dans le cas pire), on parle d'ordonnancement non-préemptif des tâches, ou (2) jusqu'à ce que la tâche en cours d'exécution ne soit plus la tâche de plus forte priorité, on parle alors d'ordonnancement préemptif. La tâche en cours d'exécution est interrompue et reprendra son exécution plus tard. En cas de conflit de priorité, le système choisit généralement de façon arbitraire la tâche à exécuter, parmi celles de plus forte priorité.

Les premiers travaux sur les systèmes de tâches temps réel remontent aux années 1970 et les travaux de Liu & Layland. Depuis, de très nombreuses contributions ont vu le jour. L'une d'elle en particulier, la méthode des trajectoires. Son étude est présentée au chapitre suivant.

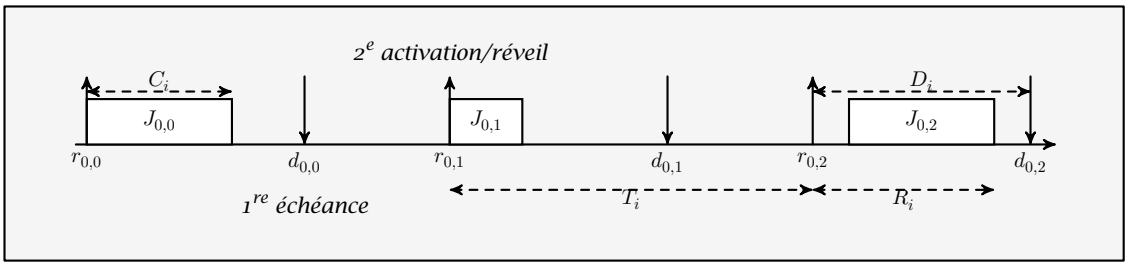

Figure 2.5 Notations : exemple pour une tâche i et trois instances.

Elle est aujourd'hui appliquée dans le secteur aéronautique pour des réseaux AFDX [9, 10] à titre de complément de la méthode utilisée actuellement pour la certification : le calcul réseau, également présenté dans le prochain chapitre.

État de l'art III

L'étude des performances en général est un domaine très vaste et plutôt mal défini. En particulier, l'étude des performances de systèmes temps réel se compose de multiples approches qui s'intéressent à différents aspects de ce que l'on nomme communément « performance ». Dans ce chapitre, nous nous positionnons au sein de ce domaine et par rapport à deux grandes familles d'approches : celles qui étudient et tentent de caractériser les comportements minimaux des systèmes et celles qui s'intéressent à leurs comportements « en moyenne ».

Des approches « prudentes » sont utilisées dans le cadre de la certification de systèmes afin de les utiliser en toute sécurité dans les domaines civils ou militaires. Ces approches garantissent un comportement et une performance minimale de ces systèmes lorsqu'ils sont embarqués à bord d'appareils (voitures, avions, satellites, etc.) ou que leur environnement est critique par exemple (barrages, centrales nucléaires, etc.). On parle aussi de performances « pire cas » en cela que ces performances caractérisent les systèmes dans les cas les plus défavorables de leurs utilisations (surcharge temporaire, conditions d'utilisation dégradées, etc.). Parmi ce type d'approches, nous nous sommes intéressés à l'étude de l'ordonnançabilité de systèmes de tâches temps réel et à celle de réseaux avioniques à qualité de service à l'aide du calcul réseau et de la méthode des trajectoires. Nous nous sommes également intéressés à la vérification formelles de propriétés à l'aide de méthodes de type *model checking*.

Les autres approches que nous avons étudiées sont utilisées dans le cadre d'estimations de comportements moyens ou approchés (comportements « les plus fréquents » ou « les plus probables »). Elles partent du constat que les situations extrêmes, et qui caractérisent les performances pire cas, sont rares dans l'absolu. Elles cherchent donc plutôt à évaluer les performances moyennes et celles observées « le plus souvent ». Parmi ces approches, nous nous sommes intéressés à l'analyse du comportement moyen de réseaux avionique à l'aide de réseaux de files d'attente, ainsi qu'à l'étude approchée du comportement réel de réseaux avioniques par la simulation guidée.

En fin de chapitre, nous nous positionnons par rapport à ces travaux et plus généralement dans le domaine de l'étude des performances des systèmes temps réel. Nous exposons

également les raisons qui ont motivé ces travaux et en quoi notre approche est originale et pertinente.

3.1 Analyses prudentes des systèmes temps réel

3.1.1 Ordonnançabilité de systèmes de tâches temps réel

L'étude de l'ordonnançabilité d'un système de tâches temps réel consiste à déterminer si chaque activation, de chacune des tâches, sera exécutée par le système avant son échéance. Ces problématiques sont couramment rencontrées dans l'industrie dès lors qu'interviennent des systèmes critiques.

Dans l'industrie automobile, il s'agit par exemple de s'assurer que les fonctions de navigation par satellite ne perturbent pas les fonctions chargées du contrôle de la trajectoire. Dans le secteur aéronautique, il s'agit entre autres de garantir que les mesures de vitesse et de température sont toujours suffisamment récentes pour les fonctions des commandes de vol.

Dans les centrales nucléaires, il faut notamment assurer un contrôle régulier de la température des cœurs des réacteurs et agir rapidement en cas de problème. Ce contrôle ne doit pas dépendre de la charge du système, sous peine de risquer un accident lorsque celui-ci effectue un grand nombre d'opérations. De même, dans un barrage, lorsqu'une alarme retentit, il faut que les vannes se referment en un délai maximal, quelque soit l'état du système à ce moment-là, sous peine d'inonder la vallée.

3.1.1.1 Tâches, instances et ordonnancement

Une tâche temps réel est un processus de calcul d'une fonction dont la correction dépend à la fois des résultats qu'elle produit et du temps nécessaire pour les calculer. Si une tâche ne respecte pas son échéance, son exécution et ses résultats sont erronés du point de vue du système.

Les instances représentent des appels à ces fonctions et sont généralement espacées régulièrement dans le temps. Elles sont au moins caractérisées par une date d'activation, une durée d'exécution et une date d'échéance ; de nombreuses autres propriétés peuvent enrichir cette base en fonction du contexte et des hypothèses. Les caractéristiques de l'instance j de la tâche i sont notées sous la forme d'un n-uplet $< r_{i,j}, C_{i,j}, d_{i,j}, \ ... >$ ou $< r_{i,j}, C_i, D_i, \ ... >$ si l'échéance relative D_i est la même pour toutes les instances ainsi que la pire durée d'exécution C_i (ou durée d'exécution maximale).

Le temps de réponse d'une instance correspond à l'intervalle de temps qui sépare son activation de la fin effective de son exécution par le système. Si l'instance reste seule dans le système durant toute son exécution, alors son temps de réponse est égal à sa durée d'exécution ; sinon la présence d'autres instances peut augmenter son temps de réponse. Conceptuellement, le temps de réponse d'une instance correspond au temps que met l'instance pour répondre à son activation. Suivant la politique d'ordonnancement du système

— qui dicte à chaque instant l'instance qui doit être exécutée en fonction de sa priorité et de celle des autres —, le temps de réponse d'une instance peut varier, notamment si deux instances de même priorité sont activées en même temps.

Une instance est ordonnançable par un système temps réel si son temps de réponse maximal possible dans ce système est inférieur à son échéance. Par suite, une tâche est ordonnançable par un système si toutes ses instances sont ordonnançables. De même, un système est ordonnançable s'il ordonnance toutes ses tâches. Le temps de réponse pire cas d'une tâche est le maximum des temps de réponse de toutes ses instances.

3.1.1.2 Les débuts de la discipline

En 1973, Liu & Layland [45] ont étudié l'ordonnançabilité des systèmes temps réel composés d'un ensemble de tâches sous les hypothèses suivantes : tâches à priorités fixes ou dynamiques, périodiques (la durée entre deux activations est constante), à échéance sur requête (l'exécution d'une instance doit être terminée avant l'activation de la suivante) et à durée d'exécution bornée par le WCET ou pire temps d'exécution de la tâche. La politique d'ordonnancement étudiée était préemptive, de type plus forte priorité d'abord. Autrement dit, l'instance qui s'exécute est toujours celle avec la priorité la plus forte, quitte à interrompre l'exécution d'une autre instance, qu'elle pourra reprendre par la suite. Ils ont démontré trois résultats importants :

- le théorème de l'instant critique : les temps de réponse des tâches sont maximaux lorsque toutes les tâches sont activées en même temps ;
- la politique d'affectation de priorité fixe, *rate monotonic*, qui affecte la priorité la plus élevée à la tâche la plus fréquente (période la plus petite) est optimale sous l'hypothèse supplémentaire de priorités fixes et si les tâches sont synchrones (première activation commune). Cela signifie que si le système est ordonnançable avec une politique d'affectation des priorités quelconque et des tâches synchrones, alors il l'est également avec *rate monotonic*. Dit autrement, cette politique d'affectation des priorités est au moins aussi efficace que toutes les autres sous les mêmes hypothèses ;
- la politique d'affectation de priorité dynamique, *earliest deadline first*, qui affecte la priorité la plus élevée à la tâche dont l'échéance est la plus proche (donc intrinsèquement dynamique) est optimale sous l'hypothèse supplémentaire de priorités dynamiques. Le résultat est même plus fort puisqu'il dit que si la charge du système est inférieure à 100 %, alors il est ordonnançable avec cette politique d'affectation.

D'autre part, ils proposent un test d'ordonnançabilité de ces systèmes en temps polynomial, basé sur une condition suffisante. Si la charge d'un système composé de n tâches vérifie l'équation (3.1), alors il est ordonnançable par *rate monotonic*.

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n \cdot \left(2^{1/n} - 1\right) \quad \xrightarrow[n\to\infty]{} 68\%. \tag{3.1}$$

Considérons un système composé de cinq tâches temps réel définies dans le tableau 3.1. Puisque les tâches sont périodiques à échéance sur requête, les dates d'activation de leurs instances sont espacées d'une durée égale à la période T de la tâche et l'échéance de toute

instance est égale à la période de la tâche (voir les notations à la section 2.5 page 15 et sur la figure 2.5). Avec ce nombre de tâches, la formule de Liu & Layland donne une charge limite égale à 74.35 % ; la charge du système est quant à elle de 100 %. Le test ne permet donc pas de conclure sur l'ordonnançabilité du système si les priorités étaient affectées par l'algorithme *rate monotonic*. *La condition est suffisante, mais pas nécessaire.*

3.1.1.3 Une littérature abondante

Depuis ces travaux fondateurs, de très nombreuses contributions ont vu le jour avec des hypothèses différentes : tâches sporadiques et apériodiques, priorité fixes et dynamiques, échéances arbitraires, multiprocesseurs et/ou multicœurs, premières activations décalées, prise en compte de gigues d'activation, partage de ressources, échange de messages à travers un *medium* de communication, prise en compte de la consommation d'énergie, etc. Nous donnons ici quelques unes des contributions majeures dans ces différents sous-domaines.

Leung & Merrill [41], dont les résultats ont été généralisés plus tard par Choquet-Geniet & Grolleau [21], ont montré qu'il existe un intervalle de temps sur lequel il possible de tester l'ordonnançabilité d'un système pour les ensembles de tâches dépendantes ou indépendantes à échéance ($D_i \leq T_i$) et pour presque tout algorithme d'ordonnancement déterministe — c'est-à-dire qui prend la même décision dans la même situation. Soit l'*hyperpériode* le plus petit commun multiple des périodes du système. Ce résultat important dit que si le système est ordonnançable sur l'intervalle $[0, \max r_i + 2 \cdot \operatorname{ppcm} T_i[$ alors il l'est partout. D'autre part, l'ordonnancement sur l'intervalle $[\max r_i + \operatorname{ppcm} T_i, \max r_i + 2 \cdot \operatorname{ppcm} T_i[$ se répète à l'infini. Par abus de language et lorsque cela ne prêtera pas à confusion, nous appelerons *hyperpériode* indifféremment le plus petit commun multiple des périodes, l'intervalle $[\max r_i + \operatorname{ppcm} T_i, \max r_i + 2 \cdot \operatorname{ppcm} T_i[$ ou tout intervalle de longueur l'hyperpériode.

Leung & Whitehead [42] ont proposé un algorithme optimal d'affectation des priorités fixes pour des ensembles de tâches préemptibles, à échéance et synchrones appelé *deadline monotonic*. Ces travaux sont une extension directe des travaux de Liu & Layland, avec D_i inférieur à T_i.

Joseph & Pandya [32] ont proposé une formule pour calculer le temps de réponse de la première instance d'une tâche pour des ensembles de tâches préemptibles, à échéance et synchrones. D'après le théorème de l'instant critique de Liu & Layland étendu par Leung & Whitehead, quelque soit les dates de réveil, les temps de réponse ne peuvent pas être plus élevés que dans le cas synchrone. Le pire temps de réponse d'une tâche se calcule alors comme le premier point fixe d'une suite récurrente définie par les équations (3.2). Ce résultat important permet de tester en temps pseudo-polynomial si un système est ordonnançable ou non. Lehockzky [40] et Sha, Rajkumar & Lehoczky [60] ont proposé des extensions pour les tâches à échéances arbitraires (D_i et T_i sans relations) et en présence de ressources partagées à l'aide d'un protocole de gestion des ressources à priorité héritée.

$$R_i^0 = B_i + C_i \qquad \text{et} \qquad R_i^{n+1} = B_i + C_i + \sum_{P_j \leq P_i} \left\lceil \frac{R_i^n}{T_j} \right\rceil \cdot C_j. \tag{3.2}$$

R_i pire temps de réponse de la tâche i (majorant)
B_i terme de blocage de la tâche i induit par les ressources partagées

C_i pire durée d'exécution de la tâche i (majorant)
T_i période de la tâche i (sépare deux instances consécutives de la tâche i)
P_i priorité de la tâche i (historiquement, une valeur plus faible indique une priorité plus forte)

L'expression $\lceil R_i^n \,/\, T_j \rceil$ correspond au nombre maximal d'activations de la tâche j dans l'intervalle entre l'activation de la tâche i à l'instant initial 0 et la fin de son exécution. Ce maximum ne prend pas en compte une activation éventuelle en R_i^n, car par définition, si l'exécution de la tâche i est terminée, la tâche j ne peut plus l'interrompre.

Mok [53] puis Jeffay & Stanat [31] ont montré que le problème de décision de l'ordonnançabilité d'un ensemble de tâches périodiques, non-préemptibles (en fait, avec la prise en compte du partage de ressources indivisibles) et concrètes (dates de réveil r_i connues) est NP-difficile et que si un algorithme en temps polynomial existait, cela impliquerait que P = NP.

Audsley [6] a proposé un algorithme optimal d'affectation des priorités fixes pour des ensembles de tâches préemptibles, périodiques, à échéances arbitraires, concrètes et asynchrones. L'idée est la suivante : l'ordonnancement étant préemptible, le temps de réponse d'une tâche ne dépend que des tâches de priorité supérieures et non de leurs priorités relatives. Ainsi, l'algorithme affecte la priorité la plus basse à une tâche puis teste si elle est ordonnançable. Si c'est le cas, l'algorithme affecte la priorité suivante, sinon, il essaie avec une autre tâche. S'il parvient à affecter une priorité à chaque tâche, le système est ordonnançable. L'algorithme d'affectation est de complexité polynomiale en $\mathcal{O}\left(n^2 + n\right)$, avec n le nombre de tâches. En revanche, le test d'ordonnançabilité nécessaire à chaque étape reste exponentiel à cause du décalage initial (tâches concrètes).

Tindell [64] puis Mäki Turja & Nolin [47], ont proposé de prendre en compte l'existence de décalages entre tâches sur certains systèmes. Par exemple, les tâches qui s'exécutent sur une partition avionique ont un décalage connu et fixe. Ils ont montré que la prise en compte de cette information améliorait sensiblement les temps de réponse pire cas calculés.

Tindell & Clark [65] ont présenté l'analyse d'ordonnançabilité holistique ou globale (preuve corrigée dans [55, 63]). Il s'agit d'étudier des systèmes temps réel distribués pour lesquels les tâches communiquent par passage de messages. Il s'agit de trouver un temps de réponse pire cas pour un ensemble de tâches distribuées. Le temps de réponse d'une tâche dans une chaîne de transactions constitue une gigue d'activation (retard de prise en compte par le système) pour la suivante. Il s'agit de majorer cette gigue afin de majorer le temps de réponse de la tâche qui en dépend. Le calcul d'une borne sur temps de réponse pire cas d'une tâche se fait ainsi de proche en proche en utilisant une suite récurrente.

$$R_i^0 = C_i \qquad \text{et} \qquad R_i^{n+1} = C_i + \sum_{P_j \leq P_i} \left\lceil \frac{R_i^n + J_j}{T_j} \right\rceil \cdot C_j. \tag{3.3}$$

J_j gigue d'activation de la tâche j égale au pire temps de réponse de la tâche sur le nœud précédent de la chaîne de transactions (voir la figure 3.1)

L'analyse d'ordonnançabilité sur des architectures multiprocesseurs et multicœurs remonte également aux années 1970. Il existe deux grandes familles d'approches : le partion-

nement et l'approche globale. La première consiste à trouver une répartions des tâches par processeur telle que sur chacun d'eux, le sous-système est ordonnançable. Les travaux fondateurs sur ce sujet sont ceux de Liu, Leung & Whitehead et Lehoczky [40, 42, 46]. Ce problème de partionnement est NP-difficile et s'apparente à ceux du *bin packing* ou du sac à dos.

L'approche globale est connue pour être NP-difficile également, sauf pour certains sous-ensembles. Baruah *et al.*[8, 62] ont proposé l'algorithme d'ordonnancement *PFair* et montré qu'il était optimal pour la classe des tâches périodiques à échéances sur requête ($D_i = T_i$). L'article de Davis & Burns sur l'ordonnancement multiprocesseurs est également une référence incontournable du domaine [24].

Enfin, le domaine des ordonnanceurs non-préemptifs est celui qui nous intéresse particulièrement dans le cadre de réseaux temps réel. Ils s'apparentent beaucoup à des ordonnanceurs avec partage de ressources où ces dernières (au moins une en fait) sont réservées du début à la fin de chaque exécution de chacune des instances. Comme déjà mentionné, Mok, Jeffay & Stanat ont montré que ce problème était NP-difficile.

En 2004, Martin [48] a proposé une extension et une amélioration de l'analyse holistique de Tindell & Clark qu'il a appelé la méthode des trajectoires. Plutôt que de tester l'ordonnançabilité d'une tâche au sein d'un système distribué de proche en proche, il propose de la vérifier sur l'ensemble de sa chaîne d'exécution. Pour cela, il caractérise le pire scénario d'arrivées et d'exécutions des tâche globalement plutôt que localement. Cela permet de grandement réduire les temps de réponse pire cas calculés de bout en bout.

3.1.1.4 La méthode des trajectoires

Dans son mémoire de thèse, Martin s'intéresse aux réseaux à qualité de service DiffServ/MPLS et propose une amélioration de la méthode holistique afin d'accepter un trafic plus important pour ces réseaux. Un trafic est accepté si toutes les trames qui transitent sur le réseau recoivent bien la qualité de service qui leur est dûe. Cela se traduit par la garantie d'une bande passante minimale pour les flux du réseau.

Du point de vue des applications qui utilisent ces trames, cela se traduit par un temps de traversée du réseau de bout en bout borné. Cette problématique est très similaire à celle des réseaux avioniques de type AFDX qui garantissent également des temps de traversée bornés, parmi d'autres propriétés et mécanismes de détection d'erreur.

Concrètement, la modélisation des trames à travers le réseau est la même que celle de Tindell & Clark. Le réseau est un système temps réel distribué où les tâches sur chacun des nœuds communiquent entre elles par échange de messages. Chaque chaîne d'exécution représente un canal de communication d'un bout à l'autre du réseau. Sur un nœud d'une chaîne, une tâche s'exécute puis envoie un message qui active la même tâche sur le nœud suivant et ainsi de suite (voir figure 3.1).

Les hypothèses que nous retenons dans ces travaux sont les suivantes : tâches périodiques — sporadiques en fait (intervalle variable borné inférieurement entre deux instances consécutives), mais cela ne change rien pour le test d'ordonnançabilité —, à priorités fixes

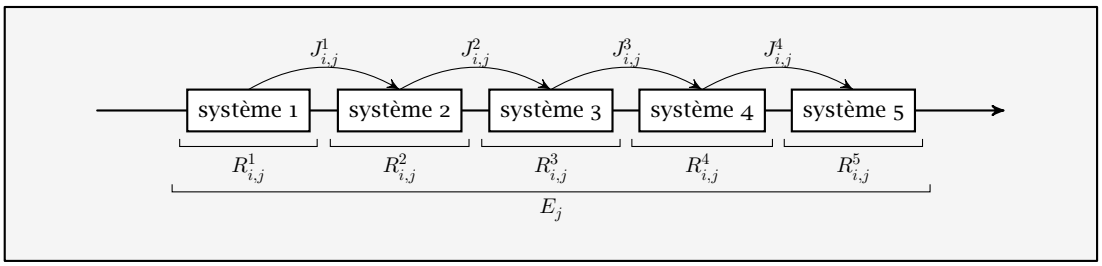

Figure 3.1 Illustration d'une ligne d'exécution.

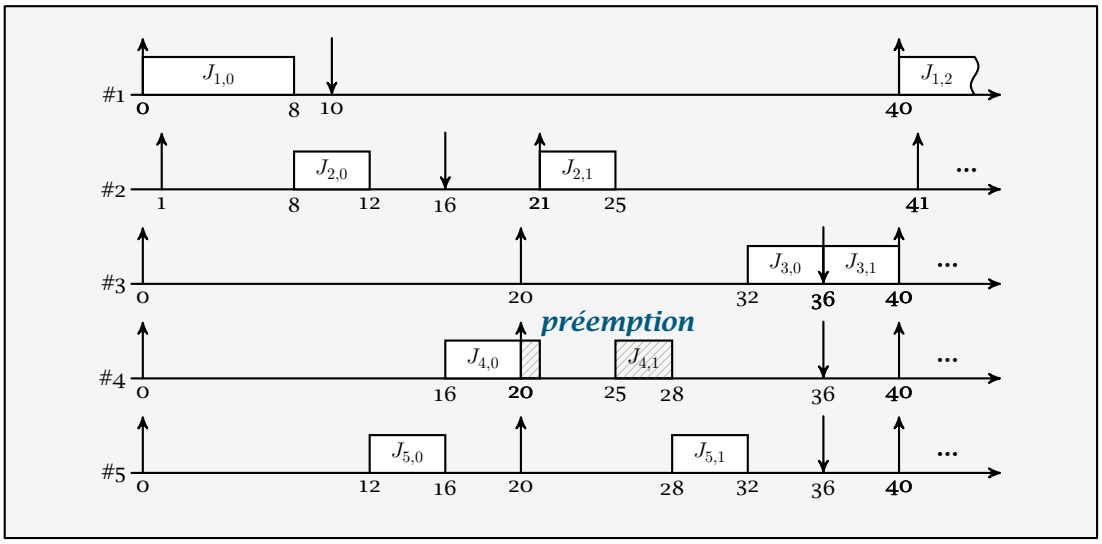

Figure 3.2 Mise en évidence de la préemption.

et non-préemptibles. Dans son mémoire, Martin propose également une version à priorités dynamiques basé sur l'algorithme d'ordonnancement et montre que celui-ci ordonnance plus de réseaux que les priorités fixes.

Les formules de calculs de pire temps de réponse en contexte préemptif prennent en compte les perturbations par des instances de priorité supérieures jusqu'à la fin de l'exécution des instances de la tâche étudiée. En contexte non-préemptif en revanche, l'exécution d'une instance ne peut être perturbée que jusqu'à un instant ϵ avant le début de celle-ci (voir les figures 3.2 et 3.3).

Concrètement, au lieu de maximiser R_i directement, il faut maximiser $W_{i,j}(t) - t$ dans l'expression $R_i(t) = W_{i,j}(t) - t + C_i$ pour toute instance j réveillée à la date t. On montre que le majorant des $W_{i,j}$ de la tâche i correspond à une instance de la tâche réveillée dans un intervalle de temps particulier et pour une configuration particulière des instants initiaux (dates des premiers réveil des tâches). Le terme $W_{i,j}$ est appelé *instant de démarrage au plus tard* de l'instance j de la tâche i, réveillée à l'intant t. Il est fonction :

- des instances plus prioritaires qui peuvent retarder l'exécution si elles arrivent jusqu'à l'instant $W_{i,j}(t)$; cet ensemble est noté hp_i ;
- des instances de même priorité qui peuvent retarder l'exécution si elles arrivent jusqu'à l'instant t puisque l'ordonnancement est FIFO par niveau de priorité ; cet ensemble est

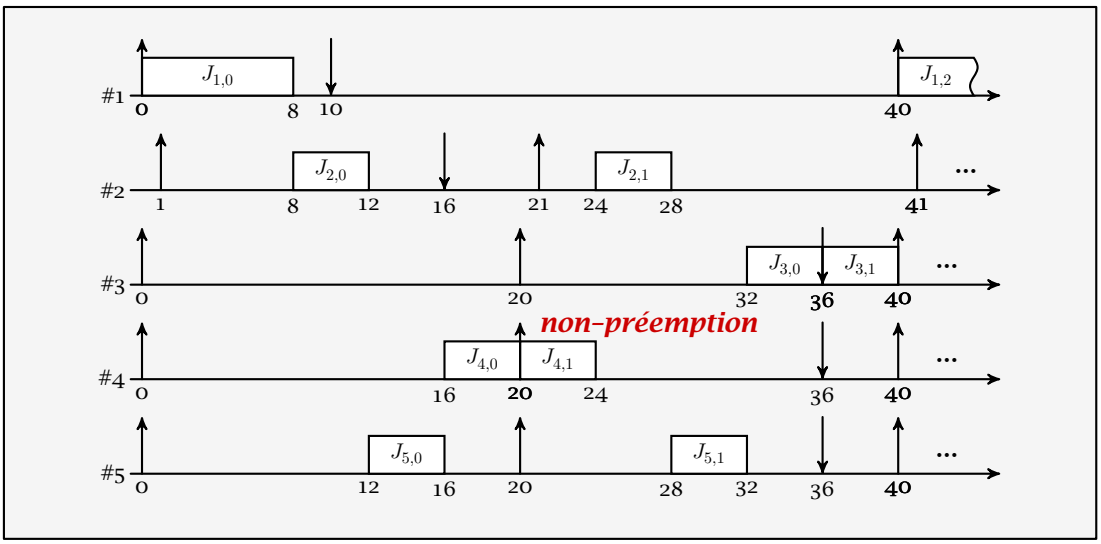

Figure 3.3 Mise en évidence de la non-préemption.

noté sp_i ;

– des instances moins prioritaires qui peuvent retarder l'exécution si elles arrivent jusqu'à l'instant précédent le début de la période active de niveau égal à celui de la tâche (voir paragraphe suivant) ; cet ensemble est noté lp_i.

D'autre part, cet instant de démarrage au plus tard apparaît lors de la plus longue période active de niveau de priorité égal à celui de la tâche étudiée. Autrement dit, lors de la plus longue durée consécutive pendant laquelle il y a dans le système des instances de priorités supérieures ou égales à celle de la tâche étudiée. On montre que cette plus longue période active pour la tâche i intervient :

– lorsque toutes les tâches de priorités supérieures ou égales démarrent le plus tôt possible ;
– mais qu'elles ne sont prises en compte qu'après une gigue d'activation maximale J_k[1] et toutes en même temps à un instant noté 0 ;
– et que de plus, la plus longue tâche la moins prioritaire s'exécute juste avant cet instant que l'on qualifie d'*initial*. Ainsi, la plus longue instances des tâches moins prioritaires que la tâche i démarre à l'intant -1.

Autrement dit, toutes les tâches de hp_i et sp_i démarrent à -J_k et sont prises en compte à 0 (*i.e.* $r_k = -J_k$ pour tout k dans hp_i et sp_i) et la plus longue tâche de lp_i démarre à -1.

À l'aide de ces notions, on peut montrer que l'instant de démarrage au plus tard s'exprime comme le plus petit point fixe de la suite récurrente $W^0_{i,j}(t) = t + \delta_i$ et

$$W^{n+1}_{i,j}(t) = \sum_{k \in hp_i} \left(1 + \left\lfloor \frac{W^n_{i,j}(t) + J_k}{T_k} \right\rfloor\right) \cdot C_k + \sum_{k \in sp_i} \left(1 + \left\lfloor \frac{t + J_k}{T_k} \right\rfloor\right) \cdot C_k + \left\lfloor \frac{t - r_i}{T_i} \right\rfloor \cdot C_i + \delta_i \qquad (3.4)$$

avec $\delta_i = \left(\max_{k \in lp_j} \{C_k\} - 1\right)$.

[1] Cette notation n'est employée que dans ce chapitre, partout ailleurs, J_j désigne l'instance j.

Dans son mémoire, Martin montre alors que le pire temps de réponse R_i de la tâche i est le maximum de l'expression $W_{i,j}(t) - t + C_i$ pour toute instance j réveillée à une date t. (La façon de réduire l'ensemble des instants t à tester dépasse le cadre de ce résumé. Pour les détails, voir [48]). De ces temps de traversée pire cas, il est théoriquement possible d'en déduire un taux d'occupation maximal des files d'attente. En pratique, la méthode n'a pas encore trouvé d'utilisation industrielle malgré de récents développements.

3.1.1.5 Illustration sur un exemple

Considérons le même exemple qu'à la section 3.1.1.2 et le système de cinq tâches décrites au tableau 3.1. Les priorités ont été affectées arbitrairement et l'exécution d'une instance est non-préemptible (colonne de droite). À titre de comparaison, nous avons calculés le pire temps de réponse en contexte préemptif à l'aide de la l'équation (3.3). Pour simplifier, les gigues d'activation sont prises nulles.

Nous pouvons constater une tendance des temps de réponse pire cas des tâches les plus prioritaires à augmenter en contexte non-préemptif par rapport au contexte préemptif. Inversement pour les tâches les moins prioritaires, les temps de réponse pire cas ont une tendance à la baisse. En effet, les priorités élevées paient le prix de la non-préemption par les priorité inférieures et ces dernières sont moins gênées par les premières car elles ne peuvent plus être préemptées en cours d'exécution.

Avec ces paramètres, le système considéré n'est pas ordonnançable par un algorithme à priorités fixes, qu'il soit préemptif ou non. Il faudrait relâcher l'échéance de la tâche 1 ou celles des tâches 3 à 5.

3.1.2 Certification d'un réseau avionique à qualité de service

Les acteurs industriels de l'aéronautique ont dû faire face à la certification de leurs réseaux, c'est-à-dire faire accepter par les autorités, une nouvelle technologie pour l'utiliser dans leurs aéronefs. Pour cela, ils se sont tournés vers des formalismes mathématiques, la rigueur et la confiance qui les accompagnent. Nous présentons ici celui qui a été retenu par Airbus pour la certification de son avion gros porteur A380.

Le calcul réseau (*Network Calculus* en anglais) est un formalisme mathématique proposé par Cruz en 1991 [22, 23], puis formalisé dans l'algèbre $(\min, +)$ par Le Boudec et Thiran

tâche	P_i	C_i	T_i	D_i	R_i (préemptif)	R_i (non-préemptif)
#1	(HP) 1	8	40	10	8	11
#2	2	4	20	15	12	15
#3	3	4	20	35	36	28
#4	3	4	20	35	36	28
#5	3	4	20	35	36	28

Tableau 3.1 Exemple d'illustration : systèmes de tâches temps réel.

en 2004 [38]. Il manipule des composants réseau et les flux qui les traversent. Les deux objectifs étaient alors de calculer des délais de traversée de bout en bout pour les flux et de borner l'utilisation des mémoires des composants afin de les dimesionner convenablement et éviter toute perte d'information par débordement de tampon (*buffer overflow*).

Les flux sont représentés par des fonctions réelles cumulatives, nulles aux temps négatifs et continues à gauche — donc nulles en zéro. Ces fonctions sont définies sur des intervalles ouverts à droite et représentent le nombre de bits « émis » par le flux entre l'instant initial et juste avant l'instant courant. Les composants sont représentés par des serveurs qui offrent des services aux flux qui les traversent.

Les bits d'un flux de fonction cumulative R qui arrivent dans un serveur S en ressortent sous la forme d'un nouveau flux de fonction cumulative R' telle que R est supérieure ou égale à R' au sens des fonctions (en tout point).

> **Remarque** Nous conseillons aux lecteurs intéressés la lecture de l'article de Boyer *et al.* [12] pour une introduction détaillée au calcul réseau. Les notations et conventions utilisées ici sont les mêmes que dans l'article. Nous ne reproduisons pas les preuves.

3.1.2.1 Courbes d'arrivée et service offert à un flux

Dans le vocabulaire du calcul réseau, les VL sont les flux de données d'un réseau AFDX ; ils sont donc modélisés par des fonctions cumulatives. Or, il est virtuellement impossible de caractériser précisément le nombre de bits qu'un VL aura émis depuis l'instant initial. En effet, les trames sont de tailles variables et dépendent de l'exécution des applications qui les produisent. D'autre part, il existe de nombreuses étapes de multiplexage tout le long des chemins des VL, au niveau des *end-systems* et des commutateurs. Néanmoins, la connaissance d'une borne maximale sur les accroissements de ces fonctions cumulatives est suffisante pour déduire de nombreux résultats intéressants.

On dit qu'une fonction α est une *courbe d'arrivée* de R, si tout accroissement de R sur un intervalle de longueur s est borné supérieurement par $\alpha(s)$. En particulier, la déconvolution de R par elle-même (ou maximisation de la différence) et définie par

$$\alpha(s) = \max_{t \geq 0} \{R(t+s) - R(t)\} \tag{3.5}$$

est une courbe d'arrivée de R, notée $R \oslash R$. C'est la courbe d'arrivée minimale de R, en ce sens que toutes courbes d'arrivée de R est supérieure ou égale à $R \oslash R$ en tout point.

Or, s'il est impossible de caractériser R directement, il est tout aussi impossible de calculer sa courbe d'arrivée minimale. Cependant, grâce aux contrats de trafic des VL, il est aisé de fournir une autre courbe d'arrivée particulière de R : le *seau percé* noté $\gamma_{r,\,b}$. Cette courbe d'arrivée correspond à un nombre de bits maximum instantané b, puis à un taux de production moyen r.

En effet, au plus $S_{\max}$ bits peuvent être émis d'un coup par un VL — ce qui correspond à une trame de taille maximale —, ensuite au plus $S_{\max}$ bits peuvent être émis tous les BAG (intervalle de temps minimal entre deux trames), soit un taux de production maximal moyen de $\frac{S_{\max}}{BAG}$. Ainsi, une autre courbe d'arrivée particulière de R est la fonction $\gamma_{S_{\max}, \frac{S_{\max}}{BAG}}$; elle est définie pour tout réel positif par l'équation

$$\alpha(s) = \begin{cases} 0 & \text{si } s < 0, \\ S_{\max} + \frac{S_{\max}}{BAG} \cdot s & \text{si } s \geq 0. \end{cases} \tag{3.6}$$

L'interprétation est la suivante : sur tout intervalle de longueur s, le VL émet au plus $\alpha(s)$ bits. Au pire, $S_{\max}$ bits sont émis instantanément au début de l'intervalle, puis en moyenne $\frac{S_{\max}}{BAG}$ bits sont émis par unité de temps pour un total de $S_{\max} + \frac{S_{\max}}{BAG} \cdot s$ au plus (avec un taux de production constant, égal au taux maximal). Graphiquement, une courbe de service de ce type est une droite affine qui « décolle » de l'axe des ordonnées à une certaine hauteur (voir la figure 3.4).

Quant à eux, les composants internes d'un réseau AFDX sont modélisés en calcul réseau par des serveurs qui offrent leurs service aux flux qui les traversent, les VL. Malheureusement, il est également virtuellement impossible de caractériser précisément le nombre de bits qu'un VL en sortie d'un nœud aura émis depuis l'instant initial. En effet, la mécanique d'ordonnancement des trames dans un nœud est complexe et dépend beaucoup du trafic. Néanmoins, la connaissance d'une borne minimale sur le nombre de bits de chaque flux que le serveur peut traiter sur tout intervalle est suffisante pour déduire de nombreux résultats intéressants.

On dit qu'un serveur offre un *service strict* de courbe β à un flux R, si tout accroissement du flux associé en sortie R' sur un intervalle de longueur s est borné inférieurement par $\beta(s)$ tant qu'il reste des bits de flux à émettre. (Par abus de langage, β est appelé la *courbe de service* offerte par le serveur à R.) Mathématiquement, pour tous instants $s \leq t$ d'une période active (intervalle ouvert à gauche tel que R y est tout point supérieure à R'), β est une courbe de service pour R si tout accroissement de R' sur un intervalle de longueur s est borné inférieurement par $\beta(s)$. En particulier,

$$\beta(s) = \min_{0 \leq t} \{R'(t+s) - R'(t)\} \tag{3.7}$$

est une courbe de service offerte par le serveur à R. C'est la courbe de service maximale qu'offre le serveur à R, toutes courbes de service de R est inférieure ou égale à celle-ci en tout point. Sinon, soit β' telle qu'il existe $\beta(s) < \beta'(s)$, alors il existe un instant t tel que $\beta'(s) > R'(t+s) - R'(t)$ et β' n'est pas une courbe de service.

De même qu'il est impossible de caractériser R' directement, il est tout aussi impossible de calculer la courbe de service maximale offerte à un flux à l'aide de cette formule. Cependant, il est possible d'exprimer une courbe de service minorante en fonction du service optimal (connu) et des courbes d'arrivées de l'ensemble des flux. En pratique, un flux reçoit un service calculé comme le service résiduel après que le serveur a servit les flux plus prioritaires. Dans un contexte « cas pire », des flux ayant la même priorité sont classés de sorte que le flux étudié soit le dernier servi.

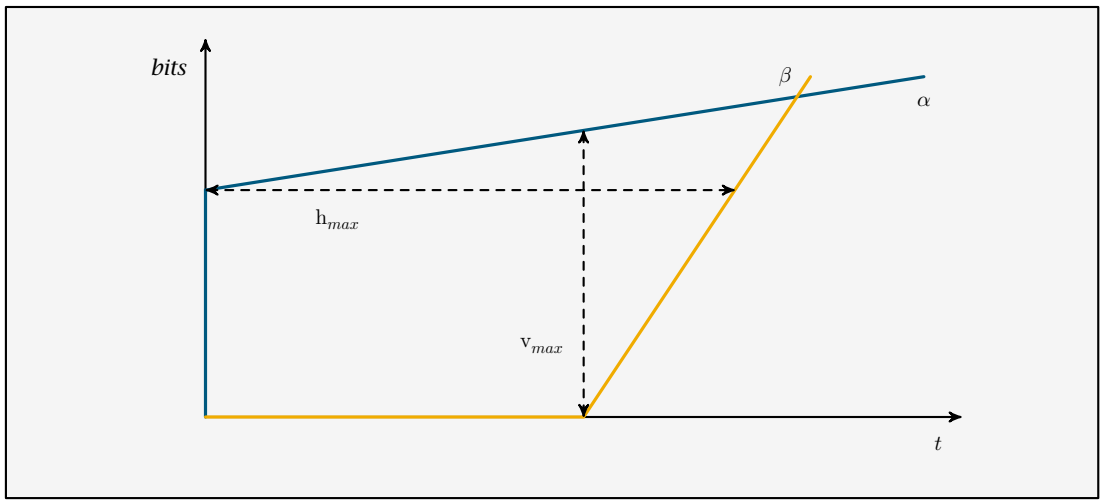

Figure 3.4 Courbes d'arrivée et de service classiques : *seau percé* et *retard*.

Dans le cas général, le service offert par un nœud d'un réseau AFDX est de type retard (la plus grosse trame des VL moins prioritaires) puis service maximal (au taux résiduel) ; il s'exprime par une relation de la forme

$$\beta_{r',d}(s) = \begin{cases} 0 & \text{si } s < 0, \\ [r' \cdot s - d]^+ & \text{si } s \geq 0. \end{cases} \tag{3.8}$$

La notation $[\bullet]^+$ signifie que les valeurs négative de $\bullet$ sont mises à zéro. Le service optimal d'un nœud est offert aux VL du niveau de priorité P_1, puis les VL du niveau de priorité P_2 reçoivent un service résiduel, et ainsi de suite. Par niveau de priorité, le traitement est FIFO. On montre que la courbe de service offerte aux VL du niveau de priorité P_i se calcule par l'équation

$$\beta^i(s) = \left[r' \cdot s - \sum_{P<P_i} \alpha(\mathrm{vl}) - \max_{P \geq P_i} S_{\max}(\mathrm{vl}) \right]^+ . \tag{3.9}$$

Graphiquement, une courbe de service de ce type est une droite affine qui « décolle » de l'axe des abscisses après un certain temps (voir la figure 3.4).

3.1.2.2 Calculs de bornes

Le dimensionnement des tampons mémoire consiste à déterminer quelle quantité minimale de mémoire il faut embarquer dans les commutateurs afin de ne pas perdre d'information par débordement de tampon (*buffer overflow*).

Ce calcul se base sur une estimation de l'accumulation maximale de bits dans un nœud en exprimant la différence entre les bits arrivés et ceux partis, puis en prenant le maximum de cette expression sur tous les intervalles. En pratique, on calcule un majorant à l'aide des courbes d'arrivée et de service par la relation

$$\mathrm{v}_{max} = \max_{s \geq 0} \{\alpha(s) - \beta(s)\} \tag{3.10}$$

et se détermine graphiquement par la plus grande distance verticale entre la courbe d'arrivée du VL et la courbe de service du nœud pour ce VL (voir la figure 3.4).

Le calcul d'une borne sur le temps maximal passé par tout bit dans un nœud (et donc par extension, par toute trame) s'exprime comme la durée maximale qui sépare les arrivées des départs. En pratique, on calcule un majorant à l'aide de l'opération de convolution des courbes d'arrivée et de service :

$$\mathrm{h}_{max} = \max_{s \geq 0} \left\{ \operatorname*{argmin}_{t \geq 0} \{\beta(s+t) - \alpha(s) \geq 0\} \right\} \tag{3.11}$$

En effet, la formule intérieure $\beta(s+t)-\alpha(s)$ représente le nombre de bits arrivés à l'instant s et encore non-envoyés après une durée t. Lorsque ce nombre est nul, tous les bits arrivés à s ont été envoyés. Donc, la fonction intérieure argmin renvoie la plus petite durée t après laquelle tous les bits arrivés à un instant s ont été envoyés. Cela correspond graphiquement à la déviation horizontale entre les deux courbes. La borne recherchée correspond au maximum de ces déviations horizontales sur tous les instants s (voir la figure 3.4 pour une interprétation graphique).

Dans le cas particulier qui nous intéresse, priorités fixes, non-préemtion et traitement FIFO par niveau de priorité, l'expression du temps de séjour maximal d'une trame d'un VL j du niveau de priorité i dans un serveur qui offre un service de type $\beta^{i}_{r',d}$ aux VL de ce niveau est de la forme[2]

$$\mathrm{h}^{i,j}_{max} = \frac{d + \sum_k b_k}{r'}. \tag{3.12}$$

b_k tel que $\alpha_k = \gamma_{r_k,b_k}$ (terme de *burst*)
d retard du serveur vu du niveau de priorité i
r' taux offert par le serveur vu du niveau de priorité i

3.1.2.3 Illustration sur un exemple

Reprenons l'exemple de la section 3.1.1.2 : cinq VL traversent un nœud de capacité unitaire. Il y a un VL de priorité élevée, un VL de priorité moyenne et trois VL de faible priorité Les résultats sont consultables dans le tableau 3.2.

VL	P_i	S_{max}	BAG	D_i	$\alpha(s)$	$\beta^i(s)$	h_{max}	R_i non-préemptif
#1	(HP) 1	8	40	10	8 + 1/5·s	$[s - 4]^+$	12	11
#2	2	4	20	15	4 + 1/5·s	$[4/5 \cdot s - 12]^+$	20	15
#3	3	4	20	35	4 + 1/5·s	$[3/5 \cdot s - 16]^+$	≈47	28
#4	3	4	20	35	4 + 1/5·s	$[3/5 \cdot s - 16]^+$	≈47	28
#5	3	4	20	35	4 + 1/5·s	$[3/5 \cdot s - 16]^+$	≈47	28

Tableau 3.2 Exemple d'illustration : calcul réseau.

[2] Voir référence [11, section 5].

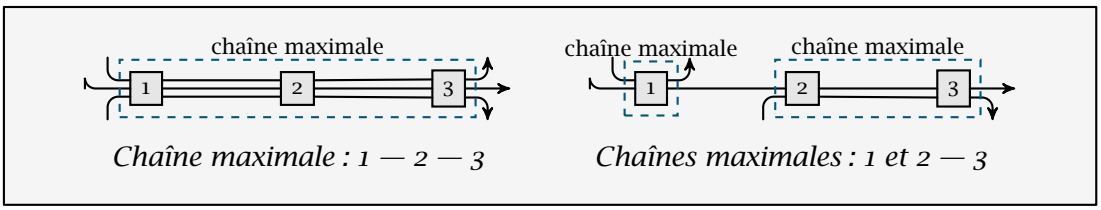

Figure 3.5 Exemple de chaîne maximale de commutateurs.

Nous pouvons remarquer que les valeurs sont généralement plus élevées que celles obtenues par la méthode des trajectoires. Attention cependant, nous n'avons pas mentionné l'intérêt majeur du calcul réseau : le phénomène appelé *pay burst only once*. Ce dernier traduit le fait que l'impact de VL qui partagent la même route ne sont ressentis qu'une seule fois le long du trajet ; les trames plus prioritaires ne repassent pas sans arrêt devant les moins prioritaires. En particulier, si deux trames de deux VL A et B traversent tout le réseau sur le même chemin et que A est plus prioritaire que B, alors la trame de A ne gênera celle de B qu'une seule fois. Une fois triées, les trames se suivent.

Cet avantage prend tout son sens lorsqu'il est possible d'extraire des enchaînements maximaux de commutateurs : c'est-à-dire tels que les VL de la chaîne n'arrivent qu'au début de celle-ci et ne la quittent qu'à la fin (voir la figure 3.5). Dans ce cas, les serveurs sont composés algébriquement.

3.1.3 Vérification de propriétés formelles par *model checking*

Le *model checking* est une technique de vérification de propriétés sur des systèmes. Les systèmes sont abstraits et modélisés sous forme d'automates (éventuellement de manière automatique) et sur lesquels il est ensuite possible de travailler. À partir d'un état initial de l'automate, celui-ci évolue à l'aide de transitions selon plusieurs critères : probabilité, condition sur une garde, temps, etc. Les états atteignables peuvent être en nombre fini ou non et il peut exister des états finals, atteints en un nombre de transition borné ou non.

À l'aide des méthodes à base de *model checking*, il est possible de vérifier qu'une propriété est vraie quelque soit l'état du système en la vérifiant pour tous les états de son automate. Il est aussi possible de vérifier qu'une propriéte est vraie sur un horizon borné, c'est-à-dire pour tout état atteignable en un nombre de transitions fixé. Dans le cas contraire, ces méthodes fournissent un contre-exemple : une suite d'états et de transitions menant à un état où la propriété n'est pas vérifiée.

Ce genre d'approches peuvent également être utilisées pour démontrer que certains états d'un système ne sont pas atteignables ou pour calculer la probabilité et le nombre de transitions nécessaire pour les atteindre. Par exemple, on peut vouloir « montrer » qu'un état fautif d'un système avionique n'est pas atteignable si certaines conditions sont remplies, ou que la probabilité pour que le système se retrouve dans un état fautif à partir de son état initial est inférieur à une certaine valeur, intervient au-delà d'un certain temps, etc.

L'utilisation du *model checking* dans sa variante temporisée a été envisagée pour la vérification de propriétés de systèmes embarqués distribués dans un cadre avionique par Carcenac *et al.* [13–15]. Pour cela, ils modélisent les ressources de communication sous la

forme de systèmes à files d'attente déterministes avec tampons de tailles bornées en utilisant le formalisme des automates temporisés (automate dans lequel les transitions se font au bout d'un certain temps). Ensuite, les vérifications peuvent être faites en utilisant des *model checkers* comme UPPAAL [5], SMV [51] ou PRISM [36] par exemple.

Par extension, il est envisageable d'utiliser des *model checkers* et leurs parcours exhaustifs d'espaces d'états pour le calcul de grandeurs dans chaque état. Après normalisation des résultats, il est ensuite possible d'en déduire une loi de distribution de ces grandeurs en fonction des états du système dans l'absolu, en prenant en compte la probabilité de chaque état pour la pondération. En particulier, il est possible d'imaginer calculer les temps de traversée pire cas d'un réseau pour tous ses VL par exemple.

En théorie, seules ces méthodes à base d'exploration exhaustive des états d'un système sont capables de fournir des résultats exacts. Dans la pratique cependant, l'explosion du nombre d'états les rend inapplicables.

3.1.4 Conclusions sur les approches prudentes

Un avantage de la méthode des trajectoires et du calcul réseau est de pouvoir caractériser les performances d'un réseau AFDX quelque soient les conditions de trafic. Cela dit, la caractérisation des performances *minimales* en est également le principal défaut car leurs occurrences sont plutôt rares dans la pratique. Selon des mesures réalisées par les avionneurs, les temps maximaux réellement observés sont de l'ordre de seulement 10 % des bornes maximales calculées. Cela découle pour beaucoup du pessimisme des méthodes employées et des modélisations plus ou moins grossières des différents éléments. De fait, ces réseaux sont plutôt surdimensionnés pour parer à quelques rares occurrences de trafic chargé.

Un autre avantage de ces deux méthodes tient en leurs complexités relativement faibles, à la fois en terme de mise en œuvre et en calcul ; cela permet en effet de considérer de grands systèmes. Cependant, les approximations qui rendent ces calculs possibles nuisent trop à la précision des résultats dès lors que le trafic du réseau augmente.

Théoriquement, les techniques de type *model checking* sont les seules capables de vérifier des propriétés et de calculer les bornes pire cas exactes de systèmes temps réel. Dans la pratique cependant, l'explosion combinatoire de l'espace d'états à explorer rend inabordable le traitement direct de systèmes réalistes. Il faut alors recourir à des abstractions qui peuvent porter préjudice à la précision, même si nombre de propriétés qualitatives restent néanmoins vérifiables. De fait, les bornes calculées par ces méthodes sont aussi pessimistes que les deux autres et pour l'heure, il n'est pas possible de calculer des bornes non pessimistes autres que pour des systèmes très simples à l'aide de ces approches.

3.2 Analyses moyennes des systèmes temps réel

3.2.1 Les réseaux de file d'attentes

3.2.1.1 Une histoire de coups de fil

Les méthodes à base de réseaux de files d'attente remontent aux travaux de Erlang au début des années 1900 [26, 27]. La problématique de l'époque était le dimensionnement suffisant des standards téléphoniques afin de minimiser les pertes d'appels entraînés par des lignes surchargées.

Erlang a proposé de modéliser les délais entre deux appels et la durée de ces derniers par des distributions dont la particularité est de ne pas tenir compte du passé. Dans l'absolu, un appel n'a pas plus de chance de se terminer parce qu'il dure depuis dix minutes que s'il dure depuis une minute ; ceci est d'autant plus vrai pour les délais entre les appels.

Cela implique que les distributions des temps d'inter-arrivée et de durée des appels sont des exponentielles décroissantes en temps continu et géométriques en temps discret. Cela signifie aussi que le nombre d'appels qui passent par le standard est un processus de Poisson. Il est alors possible de dimensionner le nombre de lignes qu'il faut prévoir pour une probabilité donnée de perte d'appels.

La propriété de perte de mémoire du temps écoulé depuis le dernier appel ou depuis le début d'un appel caractérise un processus de Markov. Il est d'ailleurs possible de représenter l'état d'un standard téléphonique à l'aide d'un système d'entrées/sorties et d'une chaîne de Markov notée M/M/K d'après les notations de Kendall : distributions des arrivées (nouveaux appels) et des départs (fin des appels) par des exponentielles décroissantes et nombre maximal d'appel K — contraint par le nombre de lignes.

La figure 3.6 illustre les différents états du système par le nombre d'appels en cours. Les termes λ et μ sont respectivement le taux des arrivées et celui des départs et sont égaux respectivement à la moyenne des arrivées et celle des départs. En effet, la moyenne d'une distribution exponentielle décroissante est égale à son taux.

La lecture de la figure est la suivante : la probabilité qu'un appel survienne alors qu'il y en a déja n en cours est égale à λ et celle que l'un des appels en cours se termine est égale à μ. En bout de chaîne, il n'est pas possible de terminer un appel s'il n'y en a plus en cours (état 0 sur la gauche), il n'est pas non plus possible d'accepter plus d'appels que de lignes (état K sur la droite).

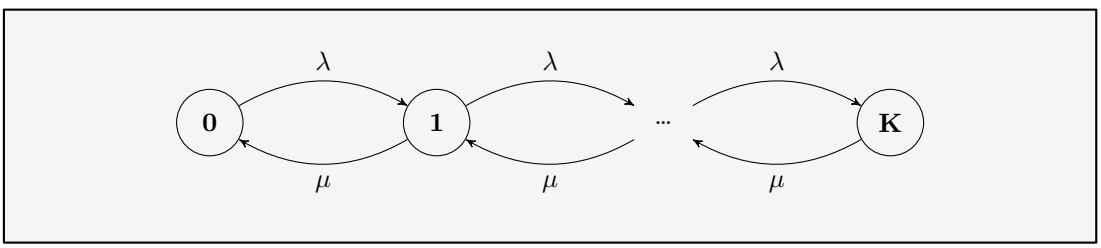

Figure 3.6 Chaîne de Markov sous-jacente d'un système M/M/K à un seul client.

Sous la condition de stabilité que le taux moyen des départs est supérieur à celui des arrivées — c'est-à-dire qu'en moyenne les appels se terminent plus vite qu'il n'en arrive —, il est possible de déduire de l'état du système la probabilité de perdre un appel parce que les lignes sont saturées. Pour le type de files d'attente M/M/K, la probabilité de perdre un appel s'exprime par le fait que le système se trouve dans l'état K et qu'un appel survient et s'exprime par l'équation

$$p(K) = \frac{\rho^K(1-\rho)}{1-\rho^{K+1}} \qquad \text{avec} \qquad \rho = \lambda \,/\, \mu. \tag{3.13}$$

3.2.1.2 Application aux réseaux AFDX

Pour appliquer cette théorie aux réseaux AFDX, on modélise l'activité dans les ports d'un réseau AFDX par des chaînes de Markov sur le modèle de client/serveur.

Un client représente un VL avec un taux de production moyen de trames, noté $\lambda_i = \frac{S_{\max}}{BAG}$, et un serveur représente la partie du port chargée d'envoyer les trames des VL à un taux moyen μ (100 $\mathrm{Mb/s}$).

Les états des files d'attente représentent le taux de remplissage dans le port, ou le nombre de trames. Chaque trame qui arrive contribue positivement au remplissage de la file (arrivée), respectivement chaque trame qui part de la file y contribue négativement (départ). Sous des hypothèses de débit moyen en sortie supérieur aux débits moyens cumulés des entrées — c'est-à-dire $\sum \lambda_i \leq \mu$ —, on peut déduire de l'état de la file la probabilité de perdre une trame parce que la file est pleine, ainsi que celle qu'une trame passe plus de temps dans le port du commutateur qu'une valeur donnée.

Dans un réseau AFDX, chaque client est caractérisé par une loi d'arrivée qui définit les instants d'arrivée des trames dans le serveur, une loi exponentielle décroissante de paramètre λ_i ; le serveur est caractérisé par un temps de service moyen qu'il offre aux clients, une loi exponentielle décroissante de paramètre μ. Le système est complété par le nombre maximal de clients dans la file (éventuellement infini). On parle de système M/M/K comme dans les travaux d'Erlang à la différence qu'ici, plusieurs clients peuvent déclencher les transitions du système à des taux différents.

Pour ce type de files d'attente, la probabilité de perdre une trame s'exprime par le fait que le système se trouve dans l'état K et qu'une trame arrive, elle s'exprime par l'équation

$$p(K) = \frac{\rho^K(1-\rho)}{1-\rho^{K+1}} \qquad \text{avec} \qquad \rho = \sum \lambda_i \,/\, \mu. \tag{3.14}$$

Il est également possible d'en déduire le temps maximal moyen qu'une trame passe dans la file d'attente.

3.2.1.3 Exemple d'illustration

Une nouvelle fois, considérons d'exemple de la section 3.1.1.2 avec une légère modification pour le client 1 — autrement, les calculs sont impossibles. Les taux d'arrivés sont égaux à

client	#1	#2	#3	#4	#5	serveur
C_i	7	4	4	5	5	
T_i	40	20	20	20	20	
λ ou μ	1/40	1/20	1/20	1/20	1/20	8/35
	0,025	0,05	0,05	0,05	0,05	$\approx$ 0,229

Tableau 3.3 Exemple d'illustration : taux des arrivées et des départs.

K (et $\rho \approx 0.984$)	1	2	3	10	20	50	100
$p(K)$ ($\approx$)	98%	49%	32%	9%	4%	1%	0,04%

Tableau 3.4 Exemple d'illustration : probabilité moyenne de perdre une trame.

l'inverse des périodes T_i dans la mesure où chaque VL produit une trame toute les périodes ; le taux du serveur est égal à la moyenne de l'inverse des tailles C_i des trames qu'il doit envoyer puisqu'il traite chaque trame *d'un* toute les *tailles de ses trames* unités de temps. Les résultats sont reportés dans le tableau 3.3. L'hypothèse débit moyen en entré inférieur au débit moyen en sortie est respectée car $9 / 40 < 8 / 35$. Si nous n'avions pas modifié l'exemple, ρ serait égal à un et donc $p(K)$ serait indéterminé.

Avec ces données, nous avons calculé la probabilité de perdre une trame en moyenne pour différentes valeurs du nombre de cases mémoire (tampon) dans le tableau 3.4. Ces résultats sont purement indicatifs dans la mesure ou la mémoire occupée ne s'exprime pas seulement en termes de cases, mais également en nombre de bits.

Ce modèle est particulièrement adapté à la modélisation d'un standard téléphonique pour lequel le temps qui sépare deux appels n'est pas connu et même totalement imprévisible, c'est-à-dire indépendant du temps des appels. Dans ce contexte les résultats sont analytiques et les calculs très rapides.

En revanche, le modèle s'avère trop simple pour des prédictions *précises* sur le comportement d'un réseau de type AFDX où les processus de génération des trames et les temps de service sont plus contraints et loin des distributions exponentielles. Néanmoins, cette approche reste intéressante en première approximation.

3.2.2 Trajectoires stochastiques

En 2007, Minet *et al.* ont proposé de combiner la méthode des trajectoires et les réseaux de files d'attente dans [52] pour en étudier les propriétés temporelles. Suivant les besoins des flux, l'une ou l'autre des approches est sélectionnées. Les flux ayant des contraintes temps réelles dures (tels que 100% des messages doivent respecter une échéance) sont traités par la méthode des trajectoires classique présentée à la section 3.1.1.4. Les flux ayant des contraintes souples (tels que la probabilité pour un message de manquer son échéance est non nulle) sont traités par la méthode dérivée des réseaux de file d'attente, brièvement

présentée ici. Ils proposent également un test d'acceptation de nouveaux flux, sans léser ceux déjà acceptés.

Les flux sont sporadiques et décrits par les paramètres habituels : période T_i, temps de traitement maximal C_i (lié à la taille maximale des messages émis par le flux), gigue d'activation J_i, échéance D_i et probabilité permise de dépasser l'échéance P_i. Pour les flux à contrainte dure, $P_i = 0$. Selon [17], les arrivées d'un flux sporadique sont bornées supérieurement par un processus de Poisson de paramètres $\lambda_i = 1 \,/\, T_i$ (taux moyen des arrivées) et $\mu_i = 1 \,/\, C_i$ (taux moyen du temps de traitement).

Le temps de réponse d'un message émis par un flux réseau est égal à la somme du temps passé dans la file d'attente et le temps de traitement, ou d'envoi, du message. La problématique du calcul de la distribution des temps de réponse, notée $S_i(t)$, est alors celle du calcul de la distribution d'une somme. Plutôt que de s'intéresser à des sommes, il est plus simple de considérer des produits car le calcul est ensuite simplifié. Pour cela, les auteurs passe du domaine temporel au domaine de Laplace par la transformée du même nom. Soit f une fonction, sa transformée de Laplace est définie par l'équation (avec les même notations, voir [52] et [34] pour les détails mathématiques)

$$(f)^*(s) = \int_0^\infty f(t) \cdot e^{-ts} dt. \tag{3.15}$$

Dans ce domaine, la (transformée de Laplace de la) distribution du temps de réponse d'un flux τ_i, notée $(S_i)^*(s)$, s'exprime comme le produit des distributions du temps d'attente dans la file et celle du temps de traitement. La formule de $(S_i)^*(s)$ étant assez complexe, le lecteur intéressé est invité à ce reporter à l'article directement. Dans le cas mono-commutateur, la propriété 5 de [52] s'exprime ainsi : un message du flux τ_i respecte son échéance D_i avec la probabilité

$$\mathcal{P}_{\text{succès}}(D_i) = \int_0^{D_i} S_i(t) dt. \tag{3.16}$$

Ainsi, un flux τ_i respecte sa contrainte souple si $\mathcal{P}_{\text{succès}}(D_i)$ est supérieur à P_i.

Les résultats présentés par les auteurs sont très intéressants et ont été validés par des simulations à l'aide de l'outil *ns2* sur un exemple simple. Par ailleurs, cette méthode prend en compte plusieurs priorités, et permet d'utiliser au mieux les ressources proposées par un réseau, même de grande taille, comme dans le deuxième exemple de l'article.

En revanche, il n'est rien dit sur le modèle de simulation utilisé, en particulier sur le modèle des flux. S'ils ont été modélisés par des processus de Poisson, il est compréhensible que les deux résultats soient si proches des simulations. D'autre part, en passant sur un exemple conséquent, seuls les résultats des simulations sont présentés. Les calculs des transformées de Laplace et des transformées inverses sont numériques, il est possible de penser que cela est un blocage suivant la taille du système considéré (cette hypothèse est cependant à confirmer auprès des auteurs). Enfin, le comportement temporel des flux est toujours modélisé par une borne supérieure et de fait, cela s'en ressentira sur la précision des résultats produits par la méthode.

3.2.3 Calcul réseau stochastique

À la même époque, Ridouard *et al.* ont proposé d'appliquer des résultats de Le Boudec et Vojnović [66] à un réseau AFDX : une extension stochastique du calcul réseau [56–58]. En particulier, ils ont étudié l'impact de flux *best effort* et l'utilisation de priorité sur les temps de traversé d'un réseau AFDX pour un groupe de VL critiques. Mais, dans un premier temps [57], ils se sont concentrés sur le cas particulier de flux traversant un seul commutateur et ayant tous la même priorité, gérée en FIFO. Selon [20], cette configuration représenterait environ 12% des cas d'une configuration de taille industrielle.

L'idée est similaire à celle de la méthode des trajectoires, mais fonctionne à l'envers. Alors que la méthode des trajectoire consiste à calculer une probabilité de ne pas dépasser une borne donnée, la méthode du calcul réseau stochastique consiste à fixer une probabilité de dépassement et à calculer la borne correspondante. Afin de calculer la loi de distribution des délais de traversée du communateur, puisque le problème est posé à l'envers, les auteurs proposent de calculer la probabilité $\mathcal{P}(d(0) > u)$ pour un paquet arrivé à un instant initial arbitraire, pour toute les valeurs de u de 1 en 1, à partir de 0.5, et tant que cette probabilité n'est pas nulle. (Plutôt, non négligeable en fait. En effet, la probabilité est bornée par une fonction décroissante de u qui tend vers zéro.) Ils en déduisent la probabilité $\mathcal{P}(d(0) = u)$, pour u entier, par l'approximation

$$\mathcal{P}(d(0) = u) = \mathcal{P}(d(0) > u - 0.5) - \mathcal{P}(d(0) > u + 0.5). \tag{3.17}$$

Cette méthode donne des résultats très intéressants. En autre, la figure 6 de l'article illustre à l'aide de valeurs calculées une croyance partagée quant aux approches prudentes et à la simulation. Les premières sont pessimistes, les secondes sont optimistes. Les résultats produits par la simulation sont décalés vers les petites valeurs des délais de traversée. Les résultats du calcul réseau sont complètement à droite. Enfin, les résultats de la méthode stochastique se situent entre les deux et les distributions décroissent très rapidement avec les valeurs des délais de traversée du commutateur.

Cependant, il s'agit encore d'une majoration et non d'une méthode de calcul précise.

3.2.4 Méthodes fluides

Dans un réseau, AFDX en particulier, un très grand nombre de trames circulent en permanence à travers les différents éléments du réseau : les nœuds terminaux (*end-systems*) et les commutateurs. Les flux s'y croisent pour être multiplexés et redirigés vers leurs destinations.

Il est difficile, voire tout à fait impossible, de suivre chaque trame de chaque flux à chaque instant dans un réeau AFDX. Dufour [25] (avec la participation de Durrieu et l'auteur lors de la phase de définition des modèles d'entrée) a eu l'idée d'adopter un point de vue eulérien plutôt que lagrangien afin d'étudier ce problème. Au–lieu de suivre chaque trame le long de sa trajectoire, il a étudié le flot des trames à travers les commutateurs du réseau et a ainsi pu estimer l'âge moyen des trames dans chacun de ces éléments.

Formellement, il a étudié l'évolution d'une fonction de distribution f dont les paramètres sont les caractéristiques des trames : l'âge, la priorité, la taille, etc. f représente la probabilité

de trouver une trame de caractéristiques c_1, c_2... dans le commutateur considéré à l'instant t. Sa loi d'évolution est déterminée à partir des évolutions des caractéristiques c_i des trames et de la somme empirique de toutes les trames selon l'équation

$$f(t, c_1, c_2, ...) = \frac{1}{N} \sum_{i=1}^{N} \delta_{c_1(t)} \delta_{c_2(t)} ... \tag{3.18}$$

Cela conduit naturellement à la résolution d'équations aux dérivées partielles dont il est possible de réduire l'espace des phases en moyennant les caractéristiques ou en ajoutant des hypothèses : priorité et taille constantes des trames au cours du temps par exemple.

3.3 Analyses simulées de réseaux AFDX

3.3.1 Principes généraux des techniques de simulation

Le domaine de la « simulation » est très varié. Il se partage en trois grandes familles :

1. la simulation pure, où tous les éléments constitutifs du système sont modélisés sur ordinateur à l'aide de modèles mathématiques ou de modèles informatiques ;
2. la simulation, où les éléments du système existent physiquement, mais dont les comportements ne sont que des approximations des éléments du système réel ;
3. la simulation partielle, quelque part entre les deux.

Assez simplement, une simulation est un modèle d'un système réel, une sortie que l'on souhaite étudier — le plus souvent, il s'agit du comportement du système par rapport à certains critères ou certaines caractéristiques en général — et un ensemble d'entrées servant à caractériser l'environnement dans lequel évoluera le système et qui influent sur ce dernier.

Les techniques de simulation sont très utilisées en phase amont de grands projets en raison essentiellement de leur faible prix, mais pas seulement. En effet, il existe de nombreuses raisons pour lesquelles une simulation est plus avantageuse qu'un essai sur du véritable matériel. Par exemple, lorsqu'une expérience est coûteuse et/ou dangereuse, à la fois pour le matériel ou pour les personnes réalisant les expériences, une simulation sur ordinateur est parfois préférable.

La conception de systèmes corrects et optimisés du premier coup est également un très bon candidat pour les activités de simulation. En anticipant les problèmes au plus tôt dans le cycle de développement des futurs systèmes, il est possible de concevoir des solutions palliatives (ou même complètement différentes), à moindre coût et cela avant même la réalisation d'un quelconque composant physique, souvent cher. Un contrôle total sur les conditions de simulation (conditions météorologiques par exemple) et la possibilité d'enregistrer et rejouer des scénarios de simulation pour analyse, bien après la simulation elle–même, sont des avantages indéniables de ce genre d'approches. Enfin, un aspect intéressant des simulations est la possibilité « d'accélérer le temps » pour évaluer le comportement à long terme des systèmes étudiés.

Malgré ces nombreux avantages, les techniques à base de simulation souffrent aussi de plusieurs inconvénients. Tout d'abord, quelque soit la richesse d'un modèle, celui–ci

n'est représentatif de la réalité que dans une certaine mesure qui ne saurait prémunir un système de tout ce qu'il pourrait rencontrer au cours de son cycle de vie. Ensuite, pour être parfaitement fidèle à l'ensemble des conditions dans lesquelles évoluera un système, il faudrait pouvoir simuler son comportement pour toute combinaisons des paramètres de la simulation (les entrées). En pratique, ces combinaisons sont bien trop nombreuses. De fait, il n'est souvent possible d'étudier qu'un sous-ensemble des conditions d'utilisation d'un système par simulation. Enfin, si une simulation représente une certaine partie du cycle de vie d'un système, comment avoir confiance par exemple, qu'un événement redouté qui aurait dû être confirmé par la simulation ne jamais pouvoir se produire, ne se manifestera effectivement pas sur le système réel ? C'est impossible à affirmer avec certitude. Une simulation n'est qu'un aperçu borné dans le temps, d'une expérience possible, vécue par le système simulé.

3.3.2 Simulation guidée de réseaux AFDX

L'objectif de la simulation guidée, proposée par Chahara *et al.* dans [16], est de calculer des bornes sur les temps de traversée de bout en bout d'un réseau AFDX moins pessimistes que celles fournies par le calcul réseau. Pour cela, ils définissent une fonction d'affinité entre les VL : plus l'affinité d'un VL avec un autre est élevé et plus celui-ci a d'impact et d'influence sur le premier, et *vice-versa*. Deux VL ayant le même chemin ont une affinité maximum, deux VL isolés n'ont aucune affinité. La caractérisation de cette affinité permet ensuite d'omettre les VL qui n'interagissent pas ou très peu avec le VL étudié afin de diminuer l'espace de simulation et rendre cette dernière abordable.

La simulation guidée se base sur une modélisation par files d'attente et l'utilisation de simulations. Dans [19], Charara propose également de classifier les trafics afin de réduire l'espace de la simulation et permettre une analyse encore plus fine du comportement moyen d'un réseau AFDX. De cette façon, la méthode permet d'évaluer une estimation du comportement moyen de chaque VL (dont le temps de traversée moyen), ainsi que le pessimisme des bornes calculées au moyen du calcul réseau ou de la méthode des trajectoires en déterminant une distribution des temps de traversée de bout en bout.

Cependant, s'agissant de simulation, il faut acquérir une confiance suffisante dans le fait que l'ensemble des scénarios de simulation retenus est effectivement représentatif et permet d'obtenir une distribution valide des temps de traversée bout en bout.

3.4 Positionnement et motivation

La problématique de l'évaluation des performances des réseaux avioniques est un domaine relativement récent où de nombreuses approches développées pour d'autre usages, sont utilisées et se combinent pour fournir différent résultats. Plus généralement, la problématique de la « performance » reste une notion vague, le domaine vaste et les interprétations nombreuses : performances minimales pour les uns, performances maximales pour les autres, performances moyennes, performances en environnement hostile, etc.

En contexte aéronautique où priment la sûreté et la sécurité, les autorités certifiantes se préoccupent en particulier des performances minimales des systèmes avioniques ; les

autres formes de performance n'ont pas ou peu d'intérêt. Cependant, nous estimons que pour mieux apprécier ces systèmes et leurs *performances*, il faut déjà en comprendre les dynamiques complexes.

Les approches prudentes ont plusieurs avantages. La méthode des trajectoires et le calcul réseau ont des complexités relativement faibles et autorisent des passages à l'échelle sans difficultés majeures. Les bornes que calculent ces méthodes sont des sur-approximations des bornes pire cas réelles, ce qui permet de garantir un fonctionnement minimal de ces systèmes. De fait, ces méthodes sont utilisées aujourd'hui pour certifier différents appareils, chez Airbus notamment ([9, 10, 30]). Enfin, la vérification de certaines propriétés formelles est possible grâce à des méthodes basées sur le *model checking*.

D'autre part, différentes approches spécialisées viennent enrichir notre connaissance des réseaux AFDX et comportent également des avantages. C'est le cas des réseaux de files d'attente qui permettent de calculer le temps de traversé de bout en bout moyen d'un réseau à l'aide d'un formalisme mathématique analytique plus ou moins adapté. La simulation guidée permet par ailleurs d'évaluer, dans une certaine mesure, le pessimisme des bornes calculées par le calcul réseau en modifiant le trafic grâce à des fonctions d'affinités, de sorte à simplifier les calculs et les simulations.

Cependant, chaque méthode est spécifique et il faut toutes les utiliser pour obtenir plusieurs résultats sur les performances d'un réseau avionique. Cela signifie plus de moyens et des coûts plus élevés. De plus, soit les approches sacrifient la précision afin de rendre les calculs possibles — calcul réseau, méthode des trajetoires et réseaux de files d'attente —, soit le coût des calculs est prohibitif à cause de problèmes d'explosion combinatoire — approches à base de *model checking*.

3.5 Notre démarche

Dans le cadre de ces travaux, nous proposons une généralisation des différentes approches présentées, ainsi qu'une manière efficace de calculer la loi de distribution complète et précise des délais de bout en bout des trames d'un flux de donnée à travers un réseau avionique de type AFDX.

Nous illustrons cette méthode en étudiant la répartition des temps de traversée maximaux d'un commutateur pour un ensemble de flux indépendants partageant le même port de sortie et la même priorité.

Tout d'abord, nous expliquons comment calculer la loi de distribution des temps de traversée du commutateur pour une trame d'un VL au chapitre V. Pour cela, nous proposons une représentation des données et un algorithme original de parcours des permutations dans la file d'attente du port de sortie du commutateur.

Puis, nous expliquons comment utiliser ces « loi de distribution locales » pour calculer la loi de distribution des temps de traversée maximaux de chaque VL au chapitre VI, pour des dates d'activation connus et fixes. Pour cela, nous introduisons la notion de trace symbolique des activations.

Enfin, nous utilisons ces calculs afin d'estimer la loi de distribution des temps de traversée maximaux d'un commutateur pour chacun des flux qui le traversent au chapitre VII. Pour cela, nous utilisons une approche statistique de type Monte Carlo.

> **Remarque** Notre approche est plutôt de type « en profondeur ». Le lecteur intéressé pourra lire les travaux particulièrement intéressants et complets de Lauer [37] pour une approche d'évaluation globale, d'application à application.

Conventions et modélisation IV

Le but de ce chapitre est de proposer une modélisation des réseaux AFDX adaptée au calcul des délais de traversée de bout en bout. Pour cela, nous rappelons et complétons les notations vues aux cours des chapitres précédents, puis nous présentons le modèle retenu pour modéliser les réseaux AFDX et enfin les hypothèses faites au cours de ces travaux.

Pour la plupart, les notations présentées ici ont été introduites aux cours des chapitres précédents. Elles empruntent volontiers au domaine des systèmes temps réel, tout en étant applicables à un cadre plus général. L'ensemble des conventions de notations utilisées dans ce manuscrit sont résumées dans le tableau 4.1 en fin de chapitre.

4.1 Conventions : vocabulaire et notations

D'une manière générale, une *tâche* représente l'exécution d'un travail *défini dans certaines conditions*. La définition du travail comprend l'ensemble des étapes nécessaires à la réalisation de la tâche et ses conditions d'exécution (ou propriétés) comme sa priorité par rapport aux autres, son échéance (durée maximale autorisée pour la réaliser), sa fréquence ou encore les ressources nécessaires à sa réalisation. Considérons un exemple : la tâche « calcul de l'assiette » peut avoir les propriétés suivantes : action/calculer l'assiette, ressources/calculateur, priorité/haute, durée/1ms, échéance/4ms et fréquence/toutes les secondes. Une échéance plus longue que la durée nécessaire pour réaliser une tâche lui permet d'être retardée.

De même, une *instance de tâche* (ou invocation) représente une réalisation particulière d'une tâche répétitive et chacune d'elle doit être exécutée correctement et en temps limité. Enfin, un *système* est un ensemble de tâches à réaliser (ou exécuter). S'il existe un ordre d'exécution de toutes les instances — on parle aussi d'ordonnancement des tâches — tel que chacune d'elles est exécutée avant son échéance, alors le système est dit *ordonnançable*.

Nous notons respectivement τ et J une tâche et une instance. L'indice i fait référence à une tâche et l'indice j à une instance ; un double indice fait référence à une instance parmi celles d'une tâche. La liste suivante recense les propriétés les plus courantes des tâches et des instances :

– ***ressources*** **tâches et instances**

  Elles peuvent être de toutes sortes et sont généralement exclusives. Par ailleurs, chaque instance reçoit l'exclusivité sur les ressources durant toute son exécution. On parle de systèmes non–préemptifs car l'exécution d'une instance ne peut être interrompue, même par d'autres instances plus prioritaires réveillées pendant son exécution.

– ***priorité*** **tâches et instances — notation P**

  Elle peut être statique ou dynamique ; on note respectivement FP (pour *Fixed Priority*) et DP (pour *Dynamic Priority*). Lorsque plusieurs instances sont en attente des ressources du système, ce dernier les affectent à l'instance avec la plus forte priorité ; on parle de politique d'ordonnancement HPF (*Highest Priority First*).

– ***durée maximale d'exécution*** **tâches et instances — notation C**

  On parle aussi de temps d'exécution « pire cas ». L'abréviation anglo–saxone WCET est largement employée (*Worst Case Execution Time*). Cette propriété est plutôt utilisée pour décrire les tâches.

– ***période*** **tâches uniquement — notation T**

  C'est la durée (respectivement durée minimale) qui sépare deux instances consécutives lorsque la tâche est périodique (respectivement sporadique). S'il n'existe pas de relation entre deux instances consécutives, la période n'est pas définie et l'on parle de tâches apériodiques (des alarmes par exemple).

– ***date de réveil (*offsets*)*** **tâches et instances — notation r**

  C'est la date à laquelle une instance signifie au système qu'elle souhaite être exécutée. Pour les tâches, cette date correspond à la date de réveil de sa première instance, *i.e.* $r_i = r_{i,0}$.

– ***échéance relative*** **tâches et instances — notation D**

  L'échéance relative d'une instance correspond à la durée maximale autorisée pour son exécution, au–delà on dit que l'instance manque son échéance. Pour une tâche, il s'agit de l'échéance relative de tous ses instances qui sont alors égales.

– ***échéance absolue*** **instances uniquement — notation d**

  C'est la date à laquelle l'exécution d'une instance doit être terminée pour qu'elle respecte son échéance. Elle est reliée à l'échéance relative et la date de réveil par la relation $d_j = r_j + D_j$. Cette propriété n'a pas de sens pour les tâches.

– ***temps de réponse*** **tâches et instances — notation R**

  C'est la durée qui sépare le réveil d'une instance et la fin de son exécution. Pour une tâche, il s'agit du maximum des temps de réponse de ses instances. Dans ce cas, on l'appelle aussi du temps de réponse pire cas de la tâche.

– ***temps de réponse de bout en bout*** **tâches et instances — notation E**

  C'est la durée qui sépare le réveil d'une instance sur le premier système d'une chaîne d'exécution et la fin de son exécution sur le dernier. Pour une tâche, il s'agit du

maximum des temps de réponse de bout en bout de ses instances. Dans ce cas, on l'appelle aussi temps de réponse pire cas de bout en bout de la tâche.

À titre d'illustration, $P_{i,j}$ désigne la priorité de la $(j+1)^{e}$ instance de la tâche i (nous comptons à partir de zéro). Si la tâche i est sporadique, alors ses instances sont liées par la relation $r_{i,j+1} - r_{i,j} \geq T_i$. Il y a (toujours) égalité lorsque la tâche est périodique et dans ce cas $r_{i,j} = r_i + j \cdot T_i$.

On définit l'instance j de la tâche i par le n–uplet $< r_{i,j}, P_{i,j}, C_{i,j}, D_{i,j} >$: date de réveil, priorité, durée maximale d'exécution, et enfin échéance relative. Si certaines propriétés sont communes pour toutes les instances d'une tâche, alors on définit la tâche aussi par un n–uplet. Par exemple, si la priorité, la durée d'exécution maximale et l'échéance relative sont communes à toutes les instances de la tâche et si, de plus, la tâche est sporadique ou périodique (*i.e.* si T_i est défini), alors on décrit la tâche i par le n–uplet $< P_i, C_i, D_i, T_i >$.

Du fait de la compétition entre les instances pour accéder aux ressources du système, les temps de réponses des instances d'une même tâche sont différents. Si le temps de réponse d'une instance dépasse son échéance, on dit que cette dernière la manque. L'étude des systèmes temps réel consiste précisément à garantir que toutes les instances d'un ensemble de tâches respecteront leurs échéances ; si c'est le cas, le système est dit ordonnançable.

4.2 Modélisation d'un réseau AFDX

Un réseau AFDX est constitué de modules (les équipements et leurs *end-systems*) et de commutateurs reliés entre eux par des câbles *Ethernet full-duplex*. Des liens virtuels, notés VL, abstraient la couche physique en un réseau de liens point à point entre les différents équipements et commutateurs. Ces liens se croisent au niveau des *nœuds* du réseau qui sont les ports d'entrée/sortie des *end-systems* et les ports de sortie des commutateurs ; ansi, il y a plusieurs nœuds par commutateur, mais un seul par *end-system* (voir la section 2.3.1).

Les trames émises par le *end-system* d'un équipement sur un VL transitent de nœud en nœud jusqu'à son ou ses *end-systems* de destination. Au niveau de chaque nœud traversé, ces trames sont en compétition avec d'autres pour accéder à la ressource *réseau* et être transmises vers le nœud suivant. Elles y sont rangées dans des files d'attente, à raison d'une file par priorité, les plus anciennes en tête. *Lorsque plusieurs trames arrivent en même temps au niveau des ports d'entrée des commutateurs, elles sont rangées aléatoirement dans leurs files de sortie respectives.* En effet, il est impossible d'enregistrer simultanément en mémoire toutes les arrivées, il faut donc faire un choix. Dans cette étude, nous avons choisi de considérer que ce choix était arbitraire. Il est possible d'envisager d'autres possibilités comme une priorité globale absolue entre tous les VL (deux VL ne peuvent donc avoir la même priorité) ou un ordre prédéterminé après le choix de la première trame. Par exemple, cela correspond à une tête de lecture qui parcours régulièrement les ports d'entrée dans un ordre précis : il est impossible de prévoir où se situe la tête au moment de l'arrivée de trames, mais l'ordre est toujours le même, modulo la première trame, etc. Dans ces nœuds, les ordonnancements suivent la politique FP/FIFO où la priorité d'une trame est plus forte qu'une autre si (1) sa priorité fixe est plus forte et (2) en cas d'égalité, si elle est « arrivée

avant » (c'est-à-dire qu'elle est placée devant dans leur file de priorité commune). Toute émission d'une trame est ensuite non-préemptible.

Ces ports, ainsi composés d'une ressource unique à partager entre un ensemble de trames récurrentes, constituent des systèmes temps réel. Nous avons choisi de les modéliser de la façon suivante :

- *liens virtuels (VL)* — tâches, notation τ
- *trames AFDX* — instances, notation J
- *nœud du réseau* — ensemble de files d'attente, une par priorité
- *nœud* + end-system — système temps réel
- *VL + nœuds + émetteurs* — réseau (de systèmes) temps réel
- *délai de traversée* — temps de réponse, notation R
- *délai de bout en bout* — temps de réponse de bout en bout, notation E

Les VL sont modélisés par des tâches et les trames par des instances ; les VL décrivent certaines propriétés des trames comme le font les tâches pour les instances. Les nœuds où se croisent les trames des différents VL, munis des émetteurs *end-systems* constituent des systèmes temps réel. Dans ce contexte, on appelle *réseau temps réel* un ensemble de systèmes temps réel reliés les uns aux autres par des relations de précédence au sens où les instances doivent s'y exécuter selon des ordres déterminés à l'avance. Dès la fin de l'exécution d'une instance sur un nœud, celle-ci est réveillée sur le nœud suivant (voir la figure 3.1 page 25). Ces nœuds sont représentés par des files d'attente dont la capacité de stockage est supposée infinie et à raison d'une file par niveau de priorité. Les trames y sont placées à leurs arrivées dans le commutateur en fonction de leurs priorités et en sont retirées après leurs émissions vers les nœuds suivants.

Le délai de traversée R_j d'une instance est la durée qui sépare son arrivée dans un nœud de son départ. D'un point de vue plus global, on parle de délai de traversée ou délai de bout en bout (*i.e.* d'un équipement à un autre) et on le note E, il est égal à la somme des délais individuels. Ces notions sont les pendants des temps de réponse et temps de réponse de bout en bout des instances dans les systèmes temps réel classiques. Si l'on étudie ces systèmes avec un point de vue « pire cas », alors E_j est inférieure, ou égale, à la somme des R_j. En effet, le temps que l'instance j arrivent au nœud k, ceux qui l'ont gênée sur les nœuds antérieurs seront peut-être déja exécutés sur celui-ci. Globalement, le délai de bout en bout pire cas est inférieur à la somme des délais pire cas sur chacun des nœuds.

4.3 Hypothèses et simplifications

Dans ce manuscrit, nous ne nous intéressons qu'aux systèmes mettant en œuvre une politique de gestion des priorités FP/FIFO non-préemptive. Toutes les notations sont regroupées dans le tableau 4.1 à la fin du chapitre. Par ailleurs, nous faisons quatres hypothèses importantes sur lesquelles nous revenons lors des conclusions de ces travaux et des perspectives envisagées.

4.3.1 Tâches indépendantes

Il s'agit d'une hypothèse classique de l'étude des systèmes et des réseaux temps réel. Deux tâches sont indépendantes s'il n'y a pas de relation de dépendance entre elles : notamment, pas de relation de précédence, pas d'échange de données, etc. Cette hypothèse simplifie les calculs et les rend abordables.

D'autre part, supposer que des tâches dépendantes ne le sont pas induit des temps de réponse égaux ou plus importants. En effet, la propriété d'indépendance implique de considérer plus d'ordonnancements qu'il n'est possible. Le temps de réponse maximal ne peut donc qu'augmenter. Il s'agit d'une propriété conservative fondamentale des méthodes de calcul de bornes pire cas pour les systèmes temps réel qui permet leurs utilisations dans des contextes de certification où il est essentiel de borner les temps d'exécution pire cas des tâches. Cependant, le conservatisme de ces méthodes se mue parfois en pessimisme lorsque les estimations sont trop éloignées des bornes réelles. Les nombreux cas impossibles en sont justement les responsables : ils sont parfois trop nombreux et « trop impossibles ».

Cette hypothèse est très forte pour les réseaux AFDX ; dans la modélisation utilisée dans cette étude, cela implique les choses suivantes :

- les *end-systems* ont autant de ports d'entrée/sortie que de VL à émettre et il y a au plus un seul VL par application sur chaque *end-system* ;
- les commutateurs ont autant de ports d'entrée que de VL qui y transitent ;
- les VL sont reconsidérés indépendants après chaque nœud traversé. Cette dernière conséquence est responsable du pessimisme des calculs et induit que la somme des bornes des délais pires cas sur chaque nœud traversé est plus grande que le délai de bout en bout en général ($\sum R_i \geq \mathrm{E}[i]$). La prise en compte de la ligne d'exécution dans son ensemble a été proposée dans [9, 48] pour les méthodes basées sur l'étude classique des systèmes temps réel. La méthode du calcul réseau prend également en compte ce phénomène de sérialisation et lui donne même un nom « *pay burst only once* ».

4.3.2 Durées d'exécution constantes

L'étude classique des systèmes temps réel suppose que le temps maximal d'exécution de chaque est tâche est connu (le WCET ou le WCTT pour les réseaux, noté C dans ce manuscrit). Il peut parfois être calculé en analysant le code source et la plate-forme d'exécution. Cependant, la plupart du temps, il est déterminé de manière empirique après plusieurs essais et/ou simulations et l'application d'une marge de sécurité. Les méthodes de calcul des bornes des temps de réponse supposent ensuite que dans la pire des situations, chaque tâche a sa durée d'exécution égale à son maximum.

Dans ce manuscrit, nous faisons la même hypothèse, sauf que l'on considère que les tâches ont toujours une durée d'exécution maximale. En terme de trame réseau, cela revient à remplir les trames plus courtes par des données vides, aisément reconnaissables pour les ignorer par la suite. Pour les systèmes temps réel, cela implique de pouvoir retarder la fin de l'exécution d'une tâche un certain temps par des instructions vides. Des études

Terme	Notation	Interprétation	
		systèmes temps réel	réseaux temps réel
ressource	—	processeur	émetteur du port de sortie
tâche	τ	« code source »	VL (lien virtuel)
système	—	tâches + ressources	VLs + nœud + émetteur
instance	J et J	« programme exécutable »	trame d'un VL
exécution d'une tâche	—	exécution d'une tâche	—
exécution d'une instance	—	exécution d'une instance	émission d'une trame
réveil d'une instance	r et r	réveil d'une instance	arrivée d'une trame
priorité (instance ou tâche)	P et P	priorité	priorité
période	T et T	période	*Bandwidth Allocation Gap*
durée max. d'exécution	C et C	*Worst Case Execution Time*	*Worst Case Transmit Time*
échéance relative	D et D	échéance relative	échéance relative
échéance absolue	d et d	échéance absolue	échéance absolue
temps de réponse	R et R	temps de réponse	délai de traversée
tps de rép. de bout en bout	E et E	tps de rép. de bout en bout	délai de bout en bout

Tableau 4.1 Notations employées dans le manuscrit.

sur les systèmes critiques dirigés par le temps s'intéressent à cette problématique, citons notamment [33, 39, 43].

4.3.3 Périodicité stricte

Nous supposons que toutes les tâches sont strictement périodiques ; autrement dit, l'intervalle de temps qui sépare deux instances consécutives d'une même tâche est strictement constant. Si cela est vrai pour les équipements de type capteurs, cela ne l'est pas en général pour les calculateurs. D'autre part, tout phénomène de dérive des horloges entre les équipements d'un réseau AFDX a été négligé. Cette hypothèse n'est pas réaliste pour des vols de plusieurs heures sans mécanisme de synchronisation des horloges. Certains protocoles ont vu le jour pour synchroniser différents équipements sur un réseau : NTP, PTP ou le projet *White Rabbit* du CERN [54, 59] par exemple.

De fait, cela exclut de considérer plus d'un nœud (commutateur) traversé par chaque VL, ou alors, les VL traversent tous les mêmes nœuds.

4.3.4 Un seul niveau de priorité

Nous n'envisageons qu'un seul niveau de priorité dans ces travaux. (Voir la section 8.2 pour des axes d'amélioration possibles.)

2

Distribution des délais de traversée d'un commutateur AFDX

Distribution d'une instance V

Dans ce chapitre, nous présentons une méthode de calcul de la loi de distribution du temps de réponse d'une instance J, activée parmi p au sein d'un système de n tâches temps réel, indépendantes, et gérées par une politique d'ordonnancement FIFO.

Pour cela, nous expliquons en quoi la répartition des instances dans la file d'attente FIFO introduit de l'indéterminisme et rend impossible un traitement direct du problème dans le cas général. Dans cette optique, nous mettons en évidence le phénomène d'explosion combinatoire et la complexité en $\mathcal{O}(n \cdot n!)$ du problème. Puis, nous introduisons la notion de « motifs » qui nous permet de représenter les données d'une manière plus compacte, de précalculer certaines informations et d'abaisser la complexité à $\mathcal{O}(n \cdot 2^n)$ dans un premier temps. Enfin, nous proposons de tirer parti de certains caractères communs entre les tâches et les instances au travers de classes d'équivalence. L'utilisation conjointe de ces classes d'équivalence et des motifs nous permet de ramener la complexité entre $\mathcal{O}(n^2)$, dans le meilleur des cas, et $\mathcal{O}(n \cdot 2^n)$, dans le plus défavorable.

5.1 Présentation du problème

Soit un commutateur un sein d'un réseau AFDX embarqué à bord d'un avion de nouvelle génération. Ce commutateur se compose de ports d'entrée/sortie où sont branchés les câbles *Ethernet*, par lesquels arrivent et repartent les trames émises sur les VL, et d'une mémoire centrale où sont stockées ces trames. Pour diverses raisons, l'écriture concurrente en mémoire est impossible. Pour régler ce problème, les industriels ont imaginé différentes solutions :

- le *round robin* : une tête de lecture parcours cycliquement les ports d'entrée, écrit en mémoire centrale les trames qu'elle y trouve et pousse un pointeur vers ces dernières dans les files d'attentes des ports de sortie concernés ;
- les ports d'entrée utilisent un bus et les trames sont prises en compte par le gestionnaire de la mémoire centrale lorsqu'elle est disponible selon un schéma similaire au précédent.

Que se passe-t-il lorsque plusieurs trames arrivent en même temps ou dans un intervalle de temps inférieur à leurs prises en compte ? Quelle trame est rangée la première dans la file (derrière celles qui y sont déjà) ? L'accès concurrent à la mémoire est indéterministe par nature et dépend d'où se trouve la tête de lecture par exemple (voir la figure 5.1).

Pour un système de tâches temps réel « classique », la conclusion est la même. (Nous rappelons que selon notre modèle, au niveau des *end-systems* et des commutateurs, le traitement d'une trame correspond au traitement d'une instance d'une tâche.) Cela est d'ailleurs à la base de l'étude d'ordonnancement de ces systèmes. Lors d'un calcul de temps de réponse pire cas d'une tâche, il est possible que toutes ses instances soient ordonnancées après toutes les autres (parmi celles qui sont en compétition directes pour le processeur). Il est donc indispensable de prendre cette possibilité en compte et de baser les calculs de temps de réponse pire cas dessus dans la mesure où cela conduit à conclure dans le « cas pire » et donc, tous les cas.

Or, à un instant donné, toutes les instances ne peuvent être ordonnancées en dernier. Aussi, il est intéressant d'étudier la loi de distribution de chaque instance devant être traitée par le système. (Si nous ne le précisons pas, nous parlons tantôt du temps de réponse et tantôt du délai, le temps de réponse moins le temps d'exécution. Ces deux notions sont tout-à-fait équivalentes dans un contexte préemptif, à la durée d'exécution près.) Dans la pratique, le problème est difficile et la combinatoire mise en jeu devient rapidement rédhibitoire. Afin de simplifier, nous supposons que l'ordre de répartition des instances dans la file d'attente FIFO est parfaitement aléatoire et uniforme ; ce n'est souvent pas le cas en pratique comme dans l'exemple du commutateur avec la tête de lecture puisque la lecture des ports d'entrée est séquentielle et cyclique. Il est cependant tout à fait possible d'en tenir compte dans l'algorithme que nous proposons dans ce manuscrit. Les résultats seraient moins pessimistes puisqu'ils ne tiendraient pas compte de situations impossibles.

Tout au long de ce chapitre, nous nous intéressons à un instant t de l'évolution d'un système de n tâches temps réel où p instances sont activées simultanément. Pour cet instant, nous souhaitons calculer la loi de distribution de chaque instance. Nous notons J une trame témoin parmi les p activées. La figure 5.1 illustre cette configuration par trois exemples d'évolutions possibles des exécutions par le système. À un certain moment après t, le système exécute les instances réveillées à t et poursuit son exécution.

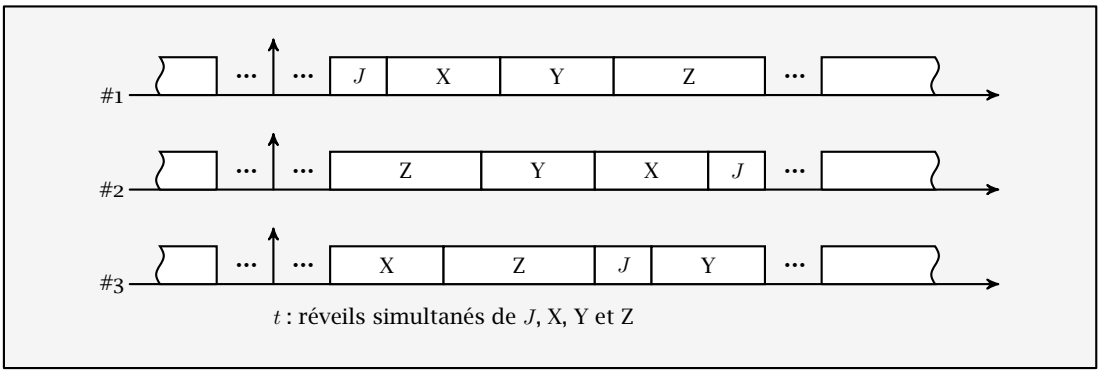

Figure 5.1 Exemples d'évolutions possibles d'un système temps réel lors de réveils simultanés.

Pour mettre en évidence le problème de l'explosion combinatoire d'une approche directe, considérons une approche naïve qui consiste à parcourir toutes les combinaisons des p instances et à calculer le temps de réponse correspondant de J. La complexité de la méthode pour l'ensemble des instances est en $\mathcal{O}(p \cdot p!)$ en temps et en espace : il faut parcourir toutes les combinaisons, sommer les temps de réponse et les stocker en mémoire pour chaque instance.

Pour chaque combinaison des p instances, il faut calculer le temps de réponse de chacune d'elle et le stocker en mémoire. Une fois que tous les temps de réponse ont été calculés, il faut compter combien de fois chaque valeur apparaît pour chaque instance et normaliser ce compte par le nombre total de combinaisons. Si en parcourant l'ensemble des combinaisons, l'instance J a q fois un temps de réponse égal à R, alors la probabilité que cette valeur soit effectivement le temps de réponse de J est égale à $q \,/\, (p!)$ dans la mesure où aucune combinaison n'est privilégiée. Ce ne serait pas le cas si nous prenions en compte l'aspect cyclique de la tête de lecture par exemple, comme évoqué précédemment.

Pour p égal à dix, $p \cdot p!$ dépasse déjà trente six millions : autant d'opérations à effectuer et de valeurs à conserver en mémoire. Dans la pratique, une telle approche n'est évidemment pas applicable à des systèmes de plusieurs dizaines, voire centaines de tâches. En effet, ces systèmes voient plusieurs fois de multiples instances activées simultanément au cours de leurs évolutions.

5.2 Introduction des motifs

Avec une approche naïve, une large part des opérations est inutilement répétée. *En effet, l'ordre des trames n'a pas d'importance pour le calcul du temps de réponse.* Sur l'exemple #3 de la figure 5.1, du point de vue de J, l'exécution de X puis Z ou Z puis X ne fait aucune différence ; seule la somme de leur temps d'exécution et de l'arriéré de travail — également appelé *backlog* — compte pour le calcul. À plus forte raison, comme sur l'exemple #1, l'ordre de l'exécution après J n'a strictement aucune importance.

Partant de ce constat, nous proposons un algorithme trivial d'une complexité plus faible en temps et en espace en $\mathcal{O}(p \cdot 2^p)$. Pour cela, notre approche consiste à ne calculer le délai engendré par un groupe d'instances qu'une seule fois, pour chaque groupe parmi 2^p. Autrement dit, nous ramenons notre calcul sur l'ensemble des parties des p instances plutôt que sur l'ensemble de leurs combinaisons. Les grandes étapes de notre algorithme sont les suivantes :

1. parcours de l'ensemble des parties des p instances ;
2. calcul du temps total d'exécution de chacune ;
3. calcul de la loi de distribution des différentes instances ;
4. normalisation des résultats avec prise en compte du *backlog* à l'instant t.

Dans la file d'attente, nous distinguons les instances qui précèdent une instance J de celles qui lui succèdent parmi celles qui sont arrivée en même temps. Nous les nommons respectivement *prédécesseurs* et *successeurs* de J. Seuls les prédécesseurs sont utiles au calcul des temps de réponse, les succésseurs n'interviennent que lors de la phase de normalisation au travers de la pondération des différentes valeurs.

Chaque instance réveillée peut être retardée dans son exécution par les instances qui se situent devant elle dans la file d'attente : celles qui y sont déjà lorsqu'elle se réveille, le *backlog*, puis celles qui sont placées avant elle parmi les instances qui se réveillent en même temps, les prédécesseurs. Laissons de côté pour l'instant les instances déjà présentes et supposons que le *backlog* est nul (cet aspect est traité au chapitre VI).

Dans cette section, nous introduisons la notion de *motif* pour représenter un groupe d'instances sans ordre relatif. Pour cela, nous considérons un exemple simple de système composé de huit tâches décrites dans le tableau 5.1. Nous utiliserons cet exemple fil rouge à travers les différents chapitres pour illustrer notre démarche. D'après l'équation (3.1) de la page 21, la charge U de ce système est légèrement supérieure à 74%. Supposons qu'à un instant quelconque t, les tâches 5, 6 et 8 génèrent chacune une instance notées respectivement J_5, J_6 et J_8.

Les différents groupes de prédecesseurs possibles de J_5 sont l'ensemble vide $\emptyset$, $\{J_6\}$, $\{J_8\}$ et $\{J_6, J_8\}$. Ceux de J_6 sont $\emptyset$, $\{J_5\}$, $\{J_8\}$ et $\{J_5, J_8\}$. Enfin, ceux de J_8 sont $\emptyset$, $\{J_5\}$, $\{J_6\}$ et $\{J_5, J_6\}$. Nous remarquons deux choses.

1. Plusieurs ensembles de prédécesseurs sont communs à plusieurs instances. C'est le cas de l'ensemble vide $\emptyset$ ou de $\{J_8\}$ par exemple. En tout, il y a douze groupes de prédécesseurs pour les trois instances, mais seulement sept sont distincts : l'ensemble vide $\emptyset$, $\{J_5\}$, $\{J_6\}$, $\{J_8\}$, $\{J_5, J_6\}$, $\{J_5, J_8\}$ et $\{J_6, J_8\}$. Nous verrons au cours de cette section et dans la suivante que ces septs groupes sont suffisants pour calculer tous les temps de réponse de chacune des instances.
2. Plusieurs groupes de prédécesseurs sont semblables au sens où ils ne diffèrent que par le numéro des instances qui les composent. Par exemple, les ensembles de prédécesseurs $\{J_5\}$ et $\{J_6\}$ d'une part, et $\{J_5, J_8\}$ et $\{J_6, J_8\}$ d'autre part, sont identiques car J_5 et J_6 sont interchangeables (car les C_i sont égaux). Cet aspect est traité à la section suivante.

5.2.1 Stratégie de parcours de l'ensemble des parties

Afin de simplifier les notations, nous choisissons de représenter les groupes de prédécesseurs par des vecteurs de taille n appelés *motifs*, n étant le nombre de tâches du système. Une valeur de 1 de la composante i indique qu'une instance de la tâche i est présente parmi les prédécesseurs et une valeur de 0 indique le contraire. Une valeur nulle peut alors avoir deux raisons : soit la tâche n'a pas généré d'instance à l'instant t, soit l'instance est parmi les successeurs. On appelle *taille du motif*, le nombre de composantes non–nulles du vecteur, c'est–à–dire la somme de ses éléments. Enfin, la *longueur du motif* correspond à la durée totale d'exécution des instances qui le composent, c'est–à–dire le produit cartésien entre le

tâche	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8
C_i	10	10	10	10	4	4	5	1
T_i	100	200	200	200	60	30	30	8

Tableau 5.1 Exemple de système temps réel.

vecteur du motif et celui des durées d'exécution des différentes tâches (supposées égales pour toutes les instances d'une même tâche, voir la section 4.3).

Illustrons cela à l'aide de l'exemple ; les motifs sont des vecteurs de taille huit. Nous avons généré une *table des motifs* au tableau 5.2. Chaque ligne représente un motif :

- la première colonne *motif* identifie le motif par un numéro ;
- les huit colonnes suivantes $\tau_1, ..., \tau_8$ constituent le motif lui-même : 1 indique la présence d'une instance de la tâche correspondante dans le motif, 0 indique une absence ;
- la colonne *taille* correspond aux nombre d'éléments dans le motif, soit dans ce cas, le nombre de composantes du motif dont la valeur est 1. Elle se calcule en sommant les composantes ;
- la dernière colonne *longueur* correspond à la durée totale d'exécution de chaque instance qui compose le motif. Elle est égale au produit cartésien entre le motif et le vecteur des durées d'exécution $\mathbf{C} = [10, 10, 10, 10, 4, 4, 5, 1]$.

Les motifs du tableau 5.2 sont présentés par taille croissante, puis par composantes de gauche à droite. Pour faire une analogie avec la représentation des nombres en base 2, cela revient à trier les nombres entre 0 et $2^8 - 1$ par nombre de bits croissants, puis par ordre décroissant des entiers naturels.

À l'aide de ces notations, les groupes distincts de prédécesseurs des instances J_5, J_6 et J_8 peuvent être regoupés selon leurs tailles :

- aucun prédécesseur : motif #1 $[0, 0, 0, 0, 0, 0, 0, 0]$, commun aux trois instances. Ce motif est utilisé pour calculer le temps de réponse d'une instance J lorsqu'elle est rangée la première dans la file d'attente parmi celles qui arrivent en même temps. Autrement dit, toutes les autres instances font parti des succésseurs de J ;
- un seul prédécesseur : motifs #6, #7 et #9, respectivement $\{J_5\}$, $\{J_6\}$ et $\{J_8\}$. Ces motifs sont utilisés pour calculer le temps de réponse d'une instance J lorsqu'elle est rangée la deuxième dans la file d'attente parmi celles qui arrivent en même temps ;
- deux prédécesseurs : motifs #32, #34 et #36, respectivement $\{J_5, J_6\}$, $\{J_5, J_8\}$ et $\{J_6, J_8\}$. Ces motifs sont utilisés pour calculer le temps de réponse d'une instance J lorsqu'elle est rangée la troisième dans la file d'attente parmi celles qui arrivent en même temps.

5.2.2 Identification des motifs

Lors du réveil d'une ou plusieurs instances, et avant de pouvoir calculer les temps de réponse et leurs lois de distribution, il faut identifier les motifs qui serviront aux calculs. Cette identification consiste en la sélection des motifs dont chaque composante est inférieure ou égale au nombre d'instances activées de la tâche correspondante — c'est-à-dire 0 ou 1 dans ce cas.

Le premier point est trivial et conduit à ne considérer que la partie supérieure de la table des motifs pour laquelle la taille des motifs est inférieure à $p - 1$, p désignant toujours le nombre d'instances réveillées à l'instant t. C'est la raison pour laquelle les motifs sont triés ainsi. *Pourquoi moins un ?* Parce que nous utilisons les motifs pour décrire les prédécesseurs d'une instance J et calculer sa loi de distribution à cet instant. Cela implique que tous les

motif	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	taille	longueur
#1	0	0	0	0	0	0	0	0	0	0
#2	1	0	0	0	0	0	0	0	1	10
#3	0	1	0	0	0	0	0	0	1	10
#4	0	0	1	0	0	0	0	0	1	10
#5	0	0	0	1	0	0	0	0	1	10
#6	0	0	0	0	1	0	0	0	1	4
#7	0	0	0	0	0	1	0	0	1	4
#8	0	0	0	0	0	0	1	0	1	5
#9	0	0	0	0	0	0	0	1	1	1
#10	1	1	0	0	0	0	0	0	2	20
#11	1	0	1	0	0	0	0	0	2	20
#12	1	0	0	1	0	0	0	0	2	20
#13	1	0	0	0	1	0	0	0	2	14
#14	1	0	0	0	0	1	0	0	2	14
...				...						...
#252	1	1	1	0	1	1	1	1	7	44
#253	1	1	0	1	1	1	1	1	7	44
#254	1	0	1	1	1	1	1	1	7	44
#255	0	1	1	1	1	1	1	1	7	44
#256	1	1	1	1	1	1	1	1	8	54

Tableau 5.2 Exemple de table des motifs.

motifs qui seront utilisés auront une taille au plus égale à $p-1$, dans la mesure où l'instance J ne fait pas parti de ses propres prédécesseurs.

Le second point requiert plus d'explications. Reprenons l'exemple de l'activation simultanée de trois instances J_5, J_6 et J_8 à un instant t. Chaque instance peut être gênée par toutes combinaisons des deux autres instances. Autrement dit, il faut considérer les motifs faisant intervenir zéro, une ou deux de celles-ci.

Pour cela, nous déterminons le *motif maître principal* : il s'agit simplement du motif correspondant à toutes les instances réveillées à cet instant. Dans notre exemple, le motif maître principal est $[0,0,0,0,1,1,0,1]$: J_5, J_6 et J_8. De ce motif de taille trois, nous savons que nous ne cherchons que des motifs de taille deux ou moins. Cela correspond aux trente-sept premières lignes de la table des motifs 5.2. En effet, il y a $\binom{0}{8} = 1$ motifs de taille zéro parmi les motifs de taille huit, $\binom{1}{8} = 8$ motifs de taille un et $\binom{2}{8} = 28$ motifs de taille deux ; au total, il y bien trente-sept motifs de taille inférieure ou égale à deux.

Ensuite, pour chaque instance, nous déterminons ce que nous appelons les *motifs maîtres secondaires*. Ces sont les mêmes motifs que le motif maître principal auquel on retire 1 à la composante correspondant à l'instance concernée. Par exemple, le motif maître secondaire de l'instance J_5 est $[0,0,0,0,\mathbf{0},1,0,1]_5$, celui de J_6 est $[0,0,0,0,1,\mathbf{0},0,1]_6$ et celui de J_8 est $[0,0,0,0,1,1,0,\mathbf{0}]_8$.

À partir d'un motif maître secondaire, nous cherchons dans la table des motifs, et parmi ceux identifiés à l'étape précédente, les *motifs esclaves* qui peuvent être des prédécesseurs de l'instance correspond au motif secondaire. Ce sont tous les motifs tels que leurs composantes sont inférieures ou égales à celle du motif secondaire de l'instance. Par exemple

- les motifs esclaves de l'instance J_5 sont
  - $[0,0,0,0,0,0,0,0]_5^{1*}$, représente les prédécesseurs $\emptyset$,
  - $[0,0,0,0,0,1,0,0]_5^{2*}$, représente les prédécesseurs $\{J_6\}$,
  - $[0,0,0,0,0,0,0,1]_5^{3*}$, représente les prédécesseurs $\{J_8\}$ et
  - $[0,0,0,0,0,1,0,1]_5^{4}$, représente les prédécesseurs $\{J_6, J_8\}$;
- les motifs esclaves de l'instance J_6 sont
  - $[0,0,0,0,0,0,0,0]_6^{1*}$, représente les prédécesseurs $\emptyset$,
  - $[0,0,0,0,1,0,0,0]_6^{2*}$, représente les prédécesseurs $\{J_5\}$,
  - $[0,0,0,0,0,0,0,1]_6^{3*}$, représente les prédécesseurs $\{J_8\}$ et
  - $[0,0,0,0,1,0,0,1]_6^{4}$, représente les prédécesseurs $\{J_5, J_8\}$;
- les motifs esclaves de l'instance J_8 sont
  - $[0,0,0,0,0,0,0,0]_8^{1*}$, représente les prédécesseurs $\emptyset$,
  - $[0,0,0,0,1,0,0,0]_8^{2*}$, représente les prédécesseurs $\{J_5\}$,
  - $[0,0,0,0,0,1,0,0]_8^{3*}$, représente les prédécesseurs $\{J_6\}$ et
  - $[0,0,0,0,1,1,0,0]_8^{4}$, représente les prédécesseurs $\{J_5, J_6\}$.

Les motifs étoilés sont communs à deux instances au moins.

5.2.3 Calcul des temps de réponse et normalisation

Une fois les motifs identifiés, il est relativement simple de calculer la ditribution du temps de réponse de chaque instance. Pour chaque instance, les différentes valeurs de temps de réponse possibles sont les longueurs des différents motifs, plus la durée d'exécution de l'instance. Le poids de chaque valeur, c'est-à-dire le nombre de combinaisons menant à ce temps de réponse, est calculé en fonction du motif esclave et du motif maître principal.

Le motif maître principal représente toutes les combinaisons des instances activée à l'instant t dans la file d'attente. Parmi ces combinaisons, chaque motif esclave d'une instance J représente les combinaisons où J est placée après les instances du motif esclave.

Il y a $k!$ permutations des instances d'un motif esclave de taille k. Ensuite, chacun des motifs esclaves est pondéré par le nombre de combinaisons induites par les permutations des instances après l'instance J, il y en a $(p-1-k)!$. Pour calculer le nombre total de combinaisons prises en compte par cette méthode, il faut prendre en compte le nombre de motifs esclaves de taille k, il y en a $\binom{k}{p-1}$. Au total, pour chaque instance, le nombre total de combinaisons est égal à

$$\sum_{k=0}^{p-1} \binom{k}{p-1} \cdot k! \cdot (p-1-k)! = p!. \tag{5.1}$$

Avec cette méthode, nous n'utilisons que 2^p motifs pour représenter $p!$ combinaisons.

instance	motif esclave	temps de réponse		poids	probabilité
J_5	$[0,0,0,0,0,0,0,0]_5^{1*}$	C_5	$= \mathbf{4}$	$0! \cdot 2! = \mathbf{2}$	**2 / 6**
	$[0,0,0,0,0,1,0,0]_5^{2*}$	$C_5 + C_6$	$= \mathbf{8}$	$1! \cdot 1! = \mathbf{1}$	**1 / 6**
	$[0,0,0,0,0,0,0,1]_5^{3*}$	$C_5 + C_8$	$= \mathbf{5}$	$1! \cdot 1! = \mathbf{1}$	**1 / 6**
	$[0,0,0,0,0,1,0,1]_5^{4}$	$C_5 + C_6 + C_8$	$= \mathbf{9}$	$2! \cdot 0! = \mathbf{2}$	**2 / 6**
J_6	$[0,0,0,0,0,0,0,0]_6^{1*}$	C_6	$= \mathbf{4}$	$0! \cdot 2! = \mathbf{2}$	**2 / 6**
	$[0,0,0,0,0,1,0,0]_6^{2*}$	$C_6 + C_5$	$= \mathbf{8}$	$1! \cdot 1! = \mathbf{1}$	**1 / 6**
	$[0,0,0,0,0,0,0,1]_6^{3*}$	$C_6 + C_8$	$= \mathbf{5}$	$1! \cdot 1! = \mathbf{1}$	**1 / 6**
	$[0,0,0,0,0,1,0,1]_6^{4}$	$C_6 + C_5 + C_8$	$= \mathbf{9}$	$2! \cdot 0! = \mathbf{2}$	**2 / 6**
J_8	$[0,0,0,0,0,0,0,0]_8^{1*}$	C_8	$= \mathbf{1}$	$0! \cdot 2! = \mathbf{2}$	**2 / 6**
	$[0,0,0,0,1,0,0,0]_8^{2*}$	$C_8 + C_5$	$= \mathbf{5}$	$1! \cdot 1! = \mathbf{1}$	**1 / 6**
	$[0,0,0,0,0,1,0,0]_8^{3*}$	$C_8 + C_6$	$= \mathbf{5}$	$1! \cdot 1! = \mathbf{1}$	**1 / 6**
	$[0,0,0,0,0,1,1,0]_8^{4}$	$C_8 + C_5 + C_6$	$= \mathbf{9}$	$2! \cdot 0! = \mathbf{2}$	**2 / 6**

Tableau 5.3 Différents temps de réponse des instances J_5, J_6 et J_8.

Par exemple, calculons les temps de réponse posibles de l'instance J_5. Le motif maître principal est $[0,0,0,0,1,1,0,1]$. Les différents temps de réponse de J_5 et leurs poids sont comme suit (les résultats pour les trois intances sont regroupés dans le tableau 5.3) :

- le motif esclave $[0,0,0,0,0,0,0,0]_5^{1*}$ entraîne un temps de réponse de J_5 égal à C_5 seulement dans la mesure où l'instance J_5 est la première dans la file d'attente. Son poids est égal au nombre de combinaisons des instances J_6 et J_8 derrière J_5 dans la file ;
- le motif esclave $[0,0,0,0,0,\mathbf{1},0,0]_5^{2*}$ entraîne un temps de réponse de J_5 égal à $C_5 + C_6$ dans la mesure où les instances sont rangées dans l'ordre J_6, puis J_5 et J_8. Son poids est égal au nombre de combinaisons de l'instance J_6 devant J_5 que multiplie le nombre de combinaisons de J_8 derrière J_5 ;
- le motif esclave $[0,0,0,0,0,0,0,\mathbf{1}]_5^{3*}$ entraîne un temps de réponse de J_5 égal à $C_5 + C_8$ dans la mesure où les instances sont rangées dans l'ordre J_8, puis J_5 et J_6. Son poids est égal au nombre de combinaisons de l'instance J_8 devant J_5 que multiplie le nombre de combinaisons de J_6 derrière J_5 ;
- le motif esclave $[0,0,0,0,0,\mathbf{1},0,\mathbf{1}]_5^{4}$ entraîne un temps de réponse de J_5 égal à $C_5+C_6+C_8$ dans la mesure où l'instance J_5 est la dernière dans la file d'attente. Son poids est égal au nombre de combinaisons des instances J_6 et J_8 devant J_5 dans la file

Une fois ces calculs faits, il ne reste plus qu'à normaliser les poids par le nombre total de combinaisons des p instances, $p!$, pour obtenir la probabilité de chaque temps de réponse. En effet, selon notre hypothèse de mise en file arbitraire, il n'y a pas de raisons que l'une des combinaisons soit privilégiée par rapport aux autres. Cela pourrait être le cas et les résultats seraient alors plus près de la vérité. Cependant, les calculs seraient également plus difficiles. Voir le tableau 5.3 pour les résultats sur l'exemple.

5.2.4 Récapitulation

Nous récapitulons ici les grandes étapes du calcul de la loi de distribution des temps de

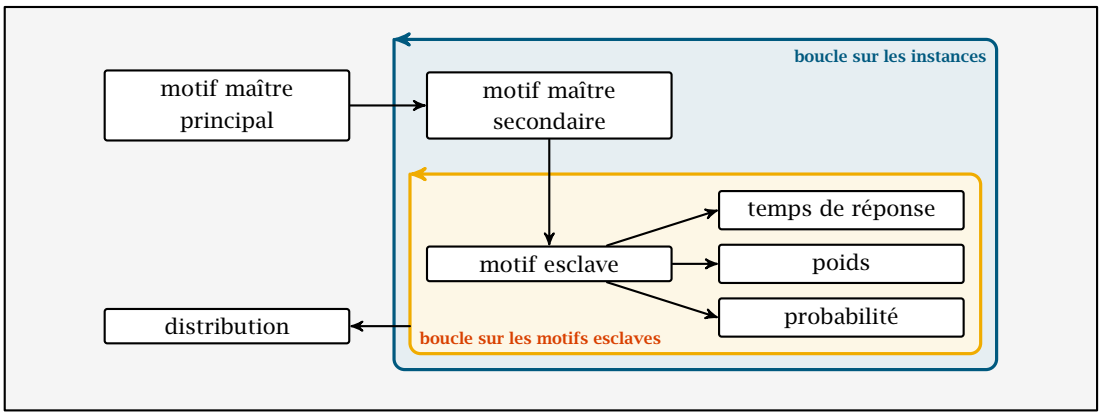

Figure 5.2 Diagramme récapitulatif du calcul des lois de distribution des temps de réponse pour un groupe d'instances.

réponse pour des instances réveillées à un instant t sans *backlog*.

1. Détermination du motif maître principal : vecteur des instances réveillées à l'instant t.
2. Pour chaque instance (boucle bleue), détermination du motif maître secondaire et des motifs esclaves : respectivement motif maître principal en soustrayant 1 à la composante correspondant à l'instance étudiée et ensembles des motifs tels que chaque composante est inférieure ou égale à celle du motif maître secondaire.
3. Pour chaque motif esclave (boucle jaune), calcul du temps de réponse de l'instance, du poids du motif esclave et normalisation pour calculer la probabilité de cette valeur.
4. Aggrégation des résultats sur tous les motifs esclaves de chaque instance afin d'obtenir leurs lois de distribution.

La complexité de l'algorithme est en $\mathcal{O}\left(p \cdot 2^p\right)$: il y a p instances et au plus 2^p motifs esclaves pour chacune d'elle. L'agrégation peut être réalisée et mise à jour tout au long des calculs des différents temps de réponse (voir [18, 35, 44, 67]).

5.3 Introduction des classes d'équivalences

L'utilisation des motifs réduit la complexité du calcul des lois de distribution d'instances reveillées simultanément. D'une complexité factorielle, nous nous ramenons à une complexité exponentielle. C'est bien, mais c'est encore insuffisant, d'autant plus que ces calculs doivent être réalisés plusieurs fois pour obtenir la loi de distribution de la tâche, c'est-à-dire pour toutes ses instances (voir le chapitre VI).

Nous proposons d'utiliser les caractères communs entre les instances et les tâches pour réduire encore la complexité. Notre idée est de rapporter la combinatoire sur les instances à une combinatoire sur des classes d'équivalence, moins nombreuses. Ces classes dépendent des données et donc la complexité ne peut être donnée que dans une fourchette : entre $\mathcal{O}\left(p \cdot 2^p\right)$ dans le cas où chaque instance est différente et $\mathcal{O}\left(p^2\right)$ dans le cas où les instances sont toutes semblables au regard de la relation d'équivalence.

Dans le domaine particulier qui nous intéresse, les réseaux avioniques AFDX, cette hypothèse revient à supposer que tous les VL ont la même taille $S_{\max}$ (ou C_i) et que toutes les

trames ont aussi toutes la même taille. À notre connaissance, aucune étude n'a à ce jour été menée sur les gains éventuels d'une telle approche, mais leurs mises en œuvre devraient être plutôt aisées.

5.3.1 Classe d'équivalence

En théorie des ensembles, la notion de *relation d'équivalence* sur un ensemble permet de mettre en relation des éléments qui sont similaires par une certaine propriété. Formellement, une relation $\mathcal{R}$ dans un ensemble $\mathcal{E}$ est une relation d'équivalence si $\mathcal{R}$ est une relation binaire réflexive, symétrique et transitive. Mathématiquement, cela s'exprime de la façon suivante :

- $\mathcal{R}$ est réflexive si $\forall x \in \mathcal{E}, x\mathcal{R}x$;
- $\mathcal{R}$ est symétrique si $\forall (x, y) \in \mathcal{E}^2, x\mathcal{R}y \Longrightarrow y\mathcal{R}x$;
- $\mathcal{R}$ est transitive si $\forall (x, y, z) \in \mathcal{E}^3, x\mathcal{R}y \wedge y\mathcal{R}z \Longrightarrow x\mathcal{R}z$.

Intuitivement, une relation d'équivalence traduit le fait que les éléments d'un ensemble ont certaines propriétés communes :

- $\mathcal{R}$ est réflexive si tout élément x de $\mathcal{E}$ est équivalent à lui–même ;
- $\mathcal{R}$ est symétrique si tout couple (x, y) d'éléments de $\mathcal{E}$ tel que x est équivalent à y implique que y est équivalent à x ;
- $\mathcal{R}$ est transitive si tout triplet (x, y, z) d'éléments de $\mathcal{E}$ tel que x est équivalent à y et y est équivalent à z implique que x est équivalent à z

Une *classe d'équivalence* d'un ensemble $\mathcal{E}$ est un ensemble d'éléments équivalents au regard d'une relation d'équivalence. Par exemple, l'ensemble des nombres pairs est une classe d'équivalence de l'ensemble des entiers naturels pour la relation *modulo* 2. La plus intuitive et la plus simple des relations d'équivalence est l'égalité.

5.3.2 Instances et tâches équivalentes

Dans le cadre de notre problématique du calcul des temps de réponse, nous avons introduit la relation d'équivalence entre tâches (respectivement instances) : deux tâches (respectivement instances) sont équivalentes si leurs durées d'exécution sont égales. Cette relation paraît naturelle au vu de la problématique.

Ainsi définie, cette relation d'équivalence partage les tâches de l'exemple en quatre sous–ensembles : $\mathbb{C}^{10}$ qui comprend les tâches de durée 10, c'est–à–dire $\{\tau_1, \tau_2, \tau_3, \tau_4\}$; $\mathbb{C}^4$ qui comprend les tâches de durée 4, c'est–à–dire $\{\tau_5, \tau_6\}$; $\mathbb{C}^5$ qui comprend les tâches de durée 5, c'est–à–dire $\{\tau_7\}$; $\mathbb{C}^1$ qui comprend les tâches de durée 1, c'est–à–dire $\{\tau_8\}$.

Il est tout à fait possible de définir d'autres relations d'équivalence en fonction du calcul effectué. Au chapitre VI, nous mentionnons une relation d'équivalence qui peut s'avérer utile lors de la génération de l'horizon des instants d'activation ou pour calculer une loi locale des distributions des temps de réponse des tâches.

5.3.3 Nouvelle définition des motifs

L'utilisation des classes d'équivalence permet de ramener le problème de combinatoire sur les tâches et les instances à une combinatoire sur les classes d'équivalence, beaucoup moins nombreuses en général. Soit un système de n tâches et k classes d'équivalence notées $\mathbb{C}^{durée}_{numéro}$. — Suivant les informations utiles, l'indice ou l'exposant pourront être omis. — Le nombre de motifs distincts — le nombre de lignes de la table des motifs — passe de 2^n à

$$\text{card}\left(\{\text{motifs}\}\right) = \prod_{m=1}^{k} \left(\text{card}\left(\mathbb{C}_m\right) + 1\right), \tag{5.2}$$

compris entre $(n+1)$ pour $k = 1$ — il n'y a qu'une seule classe d'équivalence, autrement dit toutes les tâches ont la même durée d'exécution — et 2^n pour $k = n$ — il y a n classes d'équivalence, donc une tâche par classe. Il n'y a pas d'expression plus explicite dans la mesure où cette relation dépend des données. En revanche, si les classes d'équivalence sont (à peu près) équilibrées, l'expression devient

$$\text{card}\left(\{\text{motifs}\}\right) = \left(\frac{n}{k} + 1\right)^k. \tag{5.3}$$

Désormais, au lieu de représenter un vecteur des instances activées à un instant t, les motifs représenteront le nombre d'instances de chaque classe d'équivalence activées à cet instant. Les classes sont « rangées » arbitrairement par cardinalités décroissantes. Sur l'exemple, cela signifie que si toutes les instances sont activées en même temps, le motif maître principal ne sera plus $[1,1,1,1,1,1,1,1]$, mais $[4,2,1,1]$: quatre instances de la classe $\mathbb{C}_1$, deux instances de la classe $\mathbb{C}_2$, une instance de la classe $\mathbb{C}_3$ et une instance de la classe $\mathbb{C}_4$.

Reprenons l'exemple précédent d'une activation simultanée des instances J_5, J_6 et J_8, le motif maître principal n'est plus de taille huit, mais de taille quatre (comme le nombre de classes d'équivalence). Le nouveau motif maître principal est le vecteur $[0,2,0,1]$: deux instances de la classe $\mathbb{C}_2$ (J_5 et J_6) et une instance de la classe $\mathbb{C}_4$ (J_8).

Les motifs principaux secondaires sont obtenus en ôtant 1 à la composante de la classe de l'instance concernée. Les motifs esclaves sont obtenus quant à eux de la même manière : chaque composante doit être inférieure ou égale à celle du motif maître secondaire. Le tableau 5.5 liste les différents motifs esclaves pour les différentes classes d'équivalence pour l'exemple considéré.

Sur l'exemple, nous passons de 256 lignes à 60. Tous les nouveaux motifs sont listés dans le tableau 5.4.

5.3.4 Nouveaux calculs

Comment calculer les temps de réponse et les poids associés à l'aide des classes d'équivalence ? Presque de la même façon.

1. Le calcul s'effectue par classe d'équivalence et non plus par intance réveillée. En effet, les résultats sont identiques pour tous les éléments de la même classe.
2. L'identification des motifs a été traitée à la section précédente.

motif	C_1^{10}	C_2^4	C_3^5	C_4^1	taille	longueur	motif	C_1^{10}	C_2^4	C_3^5	C_4^1	taille	longueur
#1	0	0	0	0	0	0	#31	2	1	0	1	4	25
#2	1	0	0	0	1	10	#32	2	0	1	1	4	26
#3	0	1	0	0	1	4	#33	1	2	1	0	4	23
#4	0	0	1	0	1	5	#34	1	2	0	1	4	19
#5	0	0	0	1	1	1	#35	1	1	1	1	4	20
#6	2	0	0	0	2	20	#36	0	2	1	1	4	14
#7	1	1	0	0	2	14	#37	4	1	0	0	5	44
#8	1	0	1	0	2	15	#38	4	0	1	0	5	45
#9	1	0	0	1	2	11	#39	4	0	0	1	5	41
#10	0	2	0	0	2	8	#40	3	2	0	0	5	38
#11	0	1	1	0	2	9	#41	3	1	1	0	5	39
#12	0	1	0	1	2	5	#42	3	1	0	1	5	35
#13	0	0	1	1	2	6	#43	3	0	1	1	5	36
#14	3	0	0	0	3	30	#44	2	2	1	0	5	33
#15	2	1	0	0	3	24	#45	2	2	0	1	5	29
#16	2	0	1	0	3	25	#46	2	1	1	1	5	30
#17	2	0	0	1	3	21	#47	1	2	1	1	5	24
#18	1	2	0	0	3	18	#48	4	2	0	0	6	48
#19	1	1	1	0	3	19	#49	4	1	1	0	6	49
#20	1	1	0	1	3	15	#50	4	1	0	1	6	45
#21	1	0	1	1	3	16	#51	4	0	1	1	6	46
#22	0	2	1	0	3	13	#52	3	2	1	0	6	43
#23	0	2	0	1	3	9	#53	3	2	0	1	6	39
#24	0	1	1	1	3	10	#54	3	1	1	1	6	40
#25	4	0	0	0	4	40	#55	2	2	1	1	6	34
#26	3	1	0	0	4	34	#56	4	2	1	0	7	53
#27	3	0	1	0	4	35	#57	4	2	0	1	7	49
#28	3	0	0	1	4	31	#58	4	1	1	1	7	50
#29	2	2	0	0	4	28	#59	3	2	1	1	7	44
#30	2	1	1	0	4	29	#60	4	2	1	1	8	54

Tableau 5.4 Exemple d'une table des motifs avec classes d'équivalence.

3. Le calcul à proprement parler des temps de réponse utilise le produit scalaire entre les motifs esclaves et le vecteur des durées d'exécution des classes : dans l'exemple $[10, 4, 5, 1]$.
4. Le calcul des poids doit prendre en compte la multiplicité des classes.

Soient la classe $\mathbb{C}_c$ pour laquelle on souhaite calculer le poids de l'un de ses motifs esclaves et $[p_1, \cdots, p_c, \cdots, p_k]$ son motif maître secondaire associé. La somme des p_q est égale à $p - 1$, le nombre total d'instances réveillées moins un pour le représentant de la classe $\mathbb{C}_c$. Soit le

classe	motif esclave	temps de réponse		poids	probabilité
$\mathbb{C}_2^4$	$[0,0,0,0]_4^{1^*}$	C_5	$= \mathbf{4}$	$\binom{0}{1}\cdot\binom{0}{1}\cdot 0!\cdot 2! = \mathbf{2}$	**2 / 6**
	$[0,1,0,0]_4^{2^*}$	$2\cdot C_5$	$= \mathbf{8}$	$\binom{1}{1}\cdot\binom{0}{1}\cdot 1!\cdot 1! = \mathbf{1}$	**1 / 6**
	$[0,0,0,1]_4^{3}$	$C_5 + C_8$	$= \mathbf{5}$	$\binom{0}{1}\cdot\binom{1}{1}\cdot 1!\cdot 1! = \mathbf{1}$	**1 / 6**
	$[0,1,0,1]_4^{4}$	$2\cdot C_5 + C_8$	$= \mathbf{9}$	$\binom{1}{1}\cdot\binom{1}{1}\cdot 2!\cdot 0! = \mathbf{2}$	**2 / 6**
$\mathbb{C}_4^1$	$[0,0,0,0]_1^{1^*}$	C_8	$= \mathbf{1}$	$\binom{0}{2}\cdot 0!\cdot 2! = \mathbf{2}$	**2 / 6**
	$[0,1,0,0]_1^{2^*}$	$C_8 + C_5$	$= \mathbf{5}$	$\binom{1}{2}\cdot 1!\cdot 1! = \mathbf{2}$	**2 / 6**
	$[0,2,0,0]_1^{3}$	$C_8 + 2\cdot C_5$	$= \mathbf{9}$	$\binom{2}{2}\cdot 2!\cdot 0! = \mathbf{2}$	**2 / 6**

Tableau 5.5 Différents temps de réponse des classes d'équivalence $\mathbb{C}_2$, $\mathbb{C}_3$ et $\mathbb{C}_4$.

motif esclave $[p'_1 \leq p_1, \cdots, p'_k \leq p_k]$, son poids se calcule comme suit :

$$\text{poids}\left([p'_1,\cdots,p'_k],[p_1,\cdots,p_k]\right) = \sum_{m=1}^{k}\binom{p'_m}{p_m}\cdot(p')!\cdot(p-1-p')! \tag{5.4}$$

avec p' la somme des p'_m, c'est-à-dire la taille du motif esclave, ou encore le nombre d'instances devant le représentant de la classe $\mathbb{C}_c$.

La première factorielle prend en compte toutes les permutations des p' instances devant celle de la classe $\mathbb{C}_c$. La seconde factorielle prend en compte toutes les permutations des instances derriére celle de la classe $\mathbb{C}_c$. Enfin, chaque coefficient du binôme permet de prendre en compte le choix des instances au sein de la même classe.

Par exemple, deux instances de la classe $\mathbb{C}_2$ sont activées à l'instant t avec J_8 de la classe $\mathbb{C}_4$. Le motif maître principal à cet instant est donc le vecteur $[0,2,0,1]$. Le motif maître secondaire de la classe $\mathbb{C}_2$ est $[0,1,0,1]$ et celui de la classe $\mathbb{C}_4$ est $[0,2,0,0]$.

Du point de vue de J_8, que J_5 ou J_6 soit devant elle ne change rien. On peut d'ailleurs constater que les lignes correspondant à ces deux cas de figure dans le tableau 5.3 sont désormais regroupées en une seule dans le tableau 5.5. De même, les calculs sont identiques pour les instances J_5 et J_6 car elles sont de la même classe, il suffit de les faire une fois puis de propager les résultats. Les deux sous-blocs identiques J_5 et J_6 du tableau 5.3 sont représentés par le sous-bloc unique $\mathbb{C}_2^4$ dans le tableau 5.5.

5.4 Exemple complet

Nous présentons ici un autre exemple, celui ou les huit instances sont activées en même temps. Cette situation arrive lorsque l'on fait l'hypothèse du pire cas : toutes les tâches génèrent une instance à l'instant initial, puis le plus rapidement possible, *i.e.* toute les périodes.

Nous menons le calcul de la loi de distribution des temps de réponse de l'instance J_4 à l'instant d'activation initial (instance de la tâche 4). Avec la méthode « naïve », il faudrait

environ $8! = 40\,320$ opérations pour tout calculer ; avec notre méthode, il n'en faut que 60, soit 672 fois moins.

1. **Motif maître principal**

   Puisqu'une instance de chaque tâche est activée, le motif maître principal de cet instant est le vecteur $[4, 2, 1, 1]$.

2. **Motif maître secondaire**

   Du motif maître principal, nous déduisons le motif maître secondaire pour la classe d'équivalence $\mathbb{C}_1$, celle de l'instance J_4 que nous étudions — le résultat serait le même pour les instances J_1, J_2 et J_3. Ce motif maître secondaire est le vecteur $[3, 2, 1, 1]$.

3. **Motifs esclaves**

   Du vecteur maître secondaire, nous déduisons l'ensemble des motifs esclaves pour cette classe et cet instant. Ce sont tous les motifs de taille inférieure ou égale à la taille du motif maître secondaire et dont les composantes sont elles-même inférieures ou égales deux à deux à celles de ce dernier. Il y en a quarante-huit. (On peut remarquer que le dernier motif ne sert jamais puisqu'il n'y à que n instances activées au plus, donc au plus $n - 1$ instances devant celle considérée pour le calcul.)

La loi de distribution des délais (temps de réponse moins exécution) de J_4 est reportée dans le tableau 5.6 et sur la figure 5.3. Nous remarquons qu'elle est symétrique. En effet, trouver les instances dont la somme des durées d'exécution vaut x est identique à trouver les instances dont la somme des durées d'exécution vaut le total moins x (ce sont les autres). À titre de comparaison, la figure 5.4 représente la loi de distribution pour J_4 à l'instant d'activation 200 et la figure 5.5, la distribution de la tâche 6 (de la classe $\mathbb{C}_2$) à l'instant d'activation initial.

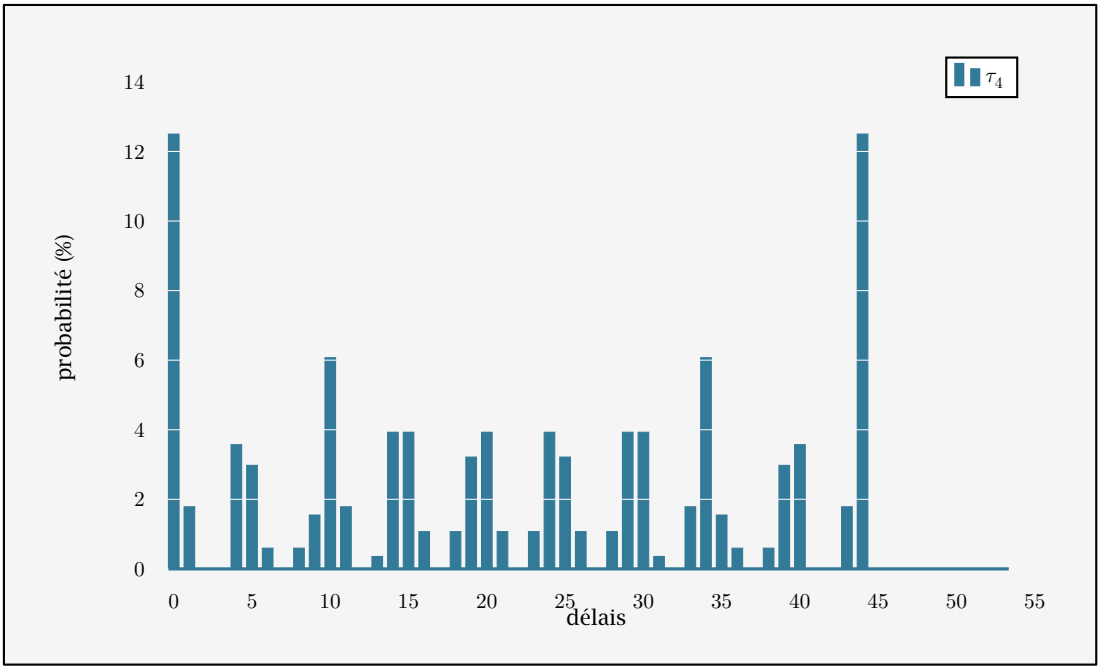

Figure 5.3 Loi de distribution des délais de la classe $\mathbb{C}_1$ à t=0.

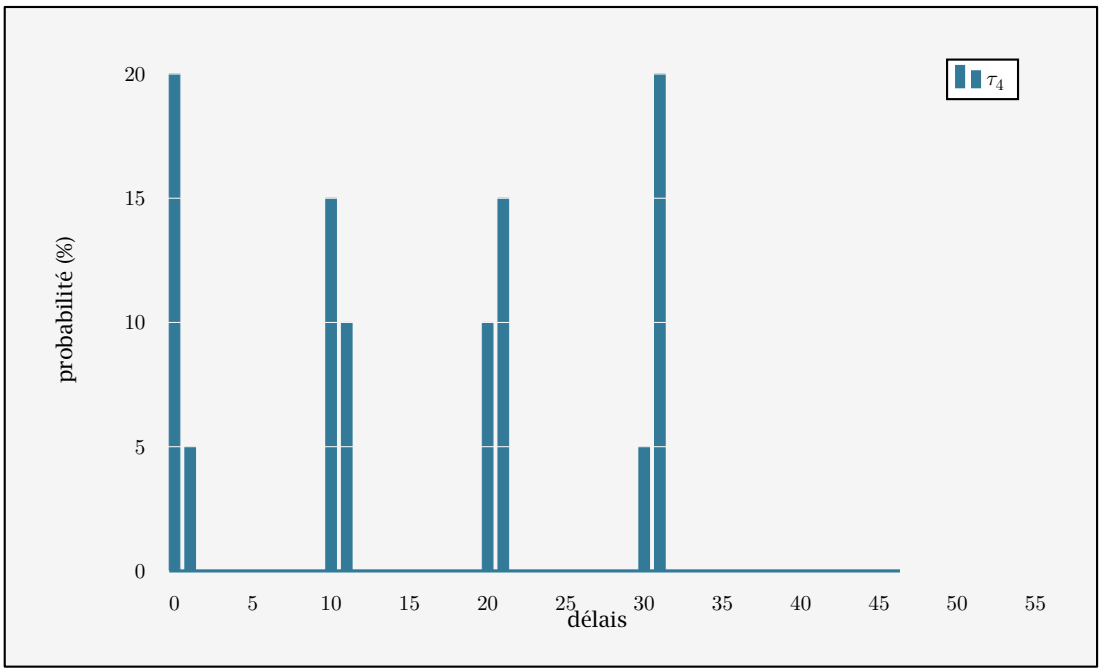

Figure 5.4 Loi de distribution des délais de la classe $\mathbb{C}_1$ à t=200.

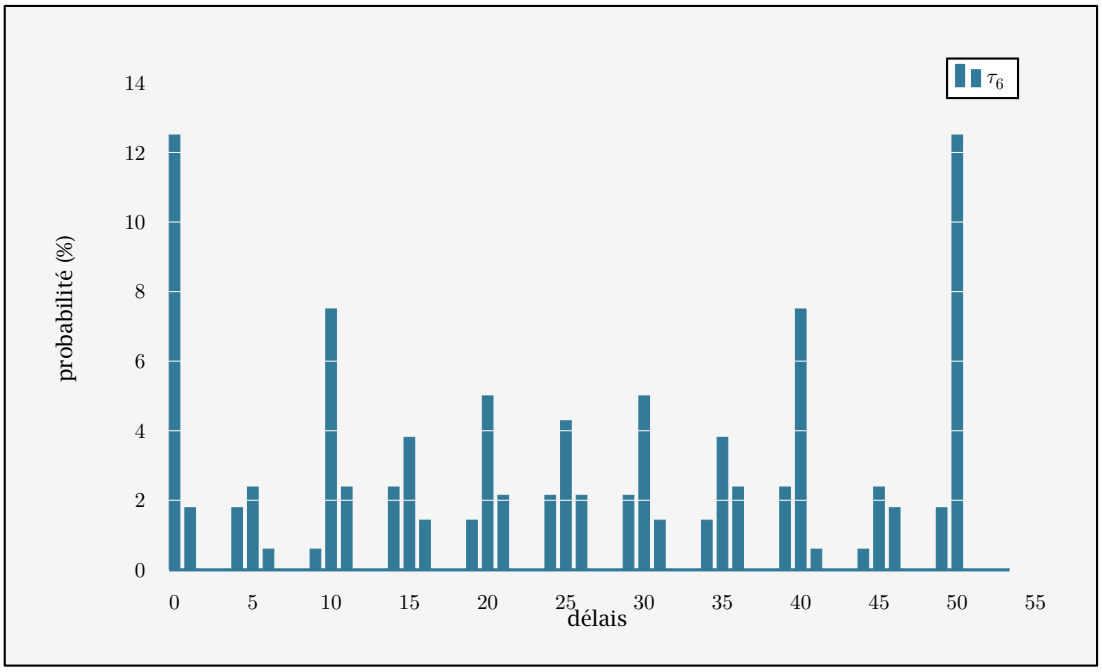

Figure 5.5 Loi de distribution des délais de la classe $\mathbb{C}_2$ à t=0.

délai	temps de réponse	motifs esclaves	poids total	probabilité (~)
0	10	#1	5040	0.125
1	11	#5	720	0.018
4	14	#3	1440	0.018
5	15	#8 et #12	1200	0.030
6	16	#13	240	0.006
8	18	#10	240	0.006
9	19	#11 et #23	624	0.015
10	20	#2 et #24	2448	0.061
11	21	#9	720	0.018
13	23	#22	144	0.004
14	24	#7 et #36	1584	0.039
15	25	#8 et #20	1584	0.039
16	26	#21	432	0.011
18	28	#18	432	0.011
19	29	#19 et #34	1296	0.032
20	30	#6 et #35	1584	0.039
21	31	#28	432	0.011
23	33	#33	432	0.011
24	34	#15 et #47	1584	0.039
25	35	#16 et #31	1296	0.032
26	36	#32	432	0.011
28	38	#40	432	0.011
29	39	#30 et #45	1584	0.039
30	40	#14 et #46	1584	0.039
31	41	#39	144	0.004
33	43	#52	720	0.018
34	44	#26 et #55	2448	0.061
35	45	#27 et #42	624	0.015
36	46	#51	240	0.006
38	48	#48	240	0.006
39	49	#41 et #53	1200	0.030
40	50	#54	1440	0.018
43	53	#52	720	0.018
44	54	#37	5040	0.125
	total	48 motifs sur 60	8! = 40320	1

Tableau 5.6 Loi de distribution des temps de réponse de la classe $\mathbb{C}_1$ à t=0.

5.5 Conclusions

Dans ce chapitre, nous avons proposé un algorithme pour le calcul des lois de distribution des temps de réponse (ou des délais) d'un ensemble d'instances réveillées simultanément au sein d'un système de tâches temps réel, monoprocesseur, non-préemptif, à priorité unique gérée en FIFO.

Pour calculer les lois de distribution, nous avons introduit deux notions : les motifs et les classes d'équivalence. Ces deux notions permettent de réduire la complexité du problème de $\mathcal{O}(n \cdot n!)$ pour une méthode naïve à $\mathcal{O}(n^2)$ au mieux et $\mathcal{O}(n \cdot 2^n)$ au pire.

La complexité dépendant des données, il n'est pas possible d'en donner une expression générale autre que l'équation (5.2). Cependant, s'il y a autant de tâches dans chaque groupe de tâches de durée d'exécution différentes — c'est-à-dire que les classes sont équilibrées —, alors une estimation de la complexité s'exprime par l'équation (5.3). Nous rappelons cette équation ci-dessous :

$$\mathrm{card}\left(\{\text{motifs}\}\right) = \prod_{m=1}^{k} \left(\mathrm{card}\left(\mathbb{C}_m\right) + 1\right) = \left(\frac{n}{k} + 1\right)^k. \tag{5.3}$$

Enfin, les motifs tels que nous les présentons sont une représentation compacte et efficace des instances dans la file d'attente, regroupées par dates de réveil commune.

Au chapitre suivant, nous utilisons cet algorithme et présentons comment prendre en compte le *backlog* et les différents instants d'activation afin de calculer les lois de distribution des temps de réponse (ou des délais) des tâches d'un système temps réel.

Distribution d'une tâche VI

Dans ce chapitre, nous présentons un algorithme efficace de calcul de la loi de distribution des temps de réponse (ou des délais) d'une tâche au sein d'un système temps réel dont les *offsets* sont connus. Pour cela, nous réutilisons les résultats du chapitre V pour le calcul des lois de distribution des instances.

Dans un premier temps, nous introduisons la notion de trace symbolique des activations. En plus d'être une notation compacte et adaptée à la problématique proposée, nous l'utilisons pour calculer certaines propriétés temporelles : le *backlog* et les temps de réponse minimaux et maximaux des tâches du système. Nous présentons ensuite notre algorithme original et le mettons en œuvre sur un exemple représentatif.

6.1 Première approche d'une trace des activations

La représentation et le stockage des structures de données sont des enjeux majeurs des méthodes exploratoires, notamment pour les méthodes à base d'automates à états finis par exemple : *model checking*, solveurs de contraintes, moteurs d'expressions régulières, etc. La raison tient au mécanisme de *backtracking* qui crée un point de retour vers un état commun pour toute branche divergente. Garder en mémoire l'ensemble des états traversés (*i.e.*, la *trace d'exécution* du système) et l'ensemble de ces points de retour (le plus souvent sous forme d'un arbre) est très coûteux, à la fois en espace mémoire et en temps. Tout ceci sans compter le temps de « calcul utile » comme la vérification de propriétés ou l'exécution de calculs numériques. Fondamentalement, une trace d'exécution est une linéarisation particulière dans le temps des états d'un système, souvent décrit de façon non–linéaire. Dans ces domaines, l'usage des BDD (*Binary Decision Diagram*) vers la fin des années 1980 a été une percée majeure.

Quelques exemples de traces d'exécution rencontrées couramment :

- une liste séquentielles des fonctions appelées lors d'un programme ;
- un fichier de *log* ;
- la pile d'erreurs d'un programme (*stack trace*) ;

- le résultat d'une simulation ;
- une suite des états parcourus par un *model checker*, une chaîne de Markov ou tout processus stochastique en général (comme les travaux d'Erlang sur les téléphones) ;
- une suite d'instructions, etc.

Une solution consiste à proposer une représentation compacte et adaptée à la problématique, ainsi qu'un algorithme de parcours efficace afin de pouvoir considérer des systèmes plus grands. Afin de mener des calculs sur les temps de réponse d'une tâches au sein d'un système temps réel, il convient d'avoir une trace d'exécution qui s'étend sur l'ensemble de l'intervalle d'étude et qui contient les informations suivantes : la date d'activation de chaque instance et sa date de fin d'exécution.

Selon nos hypothèses de périodicité stricte, nous pouvons restreindre notre étude à l'hyperpériode, c'est-à-dire l'intervalle $[\max r_i + \operatorname{ppcm} T_i, \max r_i + 2 \cdot \operatorname{ppcm} T_i[$ (voir section 3.1.1.3 et [21, 41]). De fait, nous négligeons ce qui se passe avant $\max r_i + \operatorname{ppcm} T_i$ devant la partie cyclique qui se répète à l'infini. En toute rigueur, et pour tout calcul pire cas, il faudrait en tenir compte ; en effet, le pire arrive souvent sur cet intervalle où le système « monte en charge ». D'autre part, nous stockons les dates de début d'exécution dans la mesure où nous sommes en contexte non-préemptif : la date de fin se déduit en ajoutant la durée d'exécution. Enfin, nous rappelons que l'ordre de plusieurs instances activées simultanément est supposé aléatoire (il pourrait être différent).

Évidemment en principe, ni les dates de fin, ni les dates de début d'exécution des instances ne sont connues car c'est précisément ce que l'on cherche à évaluer avec ce genre d'approches. Or, il suffit de connaître l'algorithme d'ordonnancement et les dates d'activation des différentes instances. Dans la suite du manuscrit, nous désignons par *trace des activations* toute liste de couple *date d'activation* et *instance*, ordonnés par dates croissantes.

Ci-après, trois exemples de trace des activations du système fil rouge introduit à la section 5.2. Sur les deux premiers exemples, les *offsets* sont tous nuls, seul l'ordre des instances activées à l'instant initial est différent. Dans le troisième exemple, certains *offsets* sont différents. Nous utilisons les notations $\{\bullet\}$ pour désigner une liste, $(\text{date} \mid \text{instances})$ pour désigner les « couples » date/instances et $\tau_{i,j}$ pour l'instance j de la tâche i.

1. Trace : $\{(0 \mid \tau_{1,1}), (0 \mid \tau_{2,1}), (0 \mid \tau_{3,1}), (0 \mid \tau_{4,1}), (0 \mid \tau_{5,1}), (0 \mid \tau_{6,1}), (0 \mid \tau_{7,1}), (0 \mid \tau_{8,1}), (8 \mid \tau_{8,2}), \ldots\}$.
2. Trace : $\{(0 \mid \tau_{8,1}), (0 \mid \tau_{7,1}), (0 \mid \tau_{6,1}), (0 \mid \tau_{5,1}), (0 \mid \tau_{4,1}), (0 \mid \tau_{3,1}), (0 \mid \tau_{2,1}), (0 \mid \tau_{1,1}), (8 \mid \tau_{8,2}), \ldots\}$.
3. Trace : $\{(0 \mid \tau_{8,1}), (1 \mid \tau_{1,1}), (1 \mid \tau_{2,1}), (1 \mid \tau_{3,1}), (2 \mid \tau_{4,1}), (2 \mid \tau_{5,1}), (2 \mid \tau_{6,1}), (2 \mid \tau_{7,1}), (8 \mid \tau_{8,2}), \ldots\}$.

Grâce à ces éléments, les grandes étapes d'un algorithme de calcul de la loi de distribution des tâches d'un système temps réel pourraient être les suivantes. Soit un jeu d'*offsets* donné, pour chaque instant d'activation d'au moins une tâche sur l'hyperpériode et soit p ce nombre :

1. calculer le *backlog* à cet instant (complexité en $\mathcal{O}(p)$, $p-1$ sommes) ;
2. pour chacunes des $p!$ permutations des instances activées à cet instant dans la file d'attente, calculer le temps de réponse de chaque instance nouvellement activée (complexité en $\mathcal{O}(p \cdot p!)$, voir le chapitre V) ;
3. normaliser la loi de distribution des tâches (il existe des algorithmes pour effectuer ces calculs en temps constants au fur et à mesure, soit $\mathcal{O}(1)$, voir [18, 35, 44, 67]).

Comme évoqué au chapitre V, le point 2 est particulièrement pénalisant. Une estimation grossière de la complexité globale de l'algorithme donne $\mathcal{O}\left(n \cdot n! \cdot \operatorname{ppcm} T_i\right)$, n étant le nombre de tâches du système. Le terme $\operatorname{ppcm} T_i$ surestime beaucoup le nombre d'instants d'activation sur l'intervalle hyperpériode $[\max r_i + \operatorname{ppcm} T_i, \max r_i + 2 \cdot \operatorname{ppcm} T_i[$, mais ce dernier n'est pas aisé à déterminer. Naturellement, et par souci d'équité, la même approximation sera faite aux sections suivantes.[1]

La faiblesse de cette approche (volontairement naïve pour les besoins de la démonstration) est liée au nombre des permutations des instances activées simultanément qui est en factorielle de leur nombre. Dans certaines situations, cela pourrait ne pas être un problème, s'il n'y a qu'une ou deux instances activées à chaque fois par exemple. Mais, cela ne dépend plus seulement de la taille des données, mais aussi de leurs valeurs : celles des périodes et des *offsets* en l'occurrence (voir le chapitre V).

En particulier, si les périodes sont identiques ou harmoniques (comme c'est le cas dans les réseaux AFDX où les périodes sont très souvent des puissances de deux), alors l'hyperpériode — plus petit commun multiple des périodes — est relativement faible et égale au maximum des périodes. Dans ce cas, la loi de distribution des *offsets* par rapport à cette hyperpériode est déterminante ; nous y revenons au chapitre VII.

6.2 Approche symbolique

Dans ces travaux, nous proposons de regrouper certaines instances d'une trace des activations, à la manière du chapitre précédent. Cette fois, nous regroupons les instances par date d'activation, sans tenir compte de l'ordre relatif de ces instances dans la file d'attente. Nous appelons cette nouvelle trace, la *trace symbolique des activations*.

Nous définissons la relation d'équivalence triviale « égalité » sur les jeux d'*offsets* : deux jeux sont équivalents s'ils sont égaux. Au chapitre VII, nous verrons qu'il existe une autre relation d'équivalence plus intéressante et qui permet de borner le nombre de jeux d'*offsets* à étudier. Pour l'heure, cette relation triviale est suffisante. Il est également trivial de vérifier que cette relation est bien une relation d'équivalence. Désormais, la donnée des *offsets* est suffisante pour déterminer la trace symbolique des activations sur l'ensemble de l'hyperpériode. En terme de stockage, c'est aussi plus avantageux.

Ainsi, les trois exemples donnés à la section précédente deviennent deux exemples. Les deux premiers sont équivalents (mêmes *offsets*) et sont représentés par la même trace symbolique. L'exemple 3 est représenté par une trace symbolique identique (en tous cas sur la partie montrée). Cette fois, chaque trace symbolique représente de nombreuses traces réelles d'activations des tâches ; leur nombre étant borné par à $n \cdot n! \cdot \operatorname{ppcm} T_i$).

1. Trace symbolique : $\left\{(0 \mid \tau_{1,1}, \tau_{2,1}, \tau_{3,1}, \tau_{4,1}, \tau_{5,1}, \tau_{6,1}, \tau_{7,1}, \tau_{8,1}), (8 : \tau_{8,2}), \ldots\right\}$.
2. Trace symbolique : $\left\{(0 \mid \tau_{8,1}), (1 \mid \tau_{1,1}, \tau_{2,1}, \tau_{3,1}), (2 \mid \tau_{4,1}, \tau_{5,1}, \tau_{6,1}), (3 \mid \tau_{7,1}), (8 \mid \tau_{8,2}), \ldots\right\}$.

[1] $\mathcal{O}\left(\sum_{t=1}^{n_a} p \cdot (p)!\right)$ est une meilleure approximation, avec p le nombre d'instances activée à l'instant t et n_a le nombre d'instants d'activations d'au moins une instance tel que n_a inférieur à $\sum_{j=1}^{n} \left\lceil \frac{\operatorname{ppcm} T_i}{T_j} \right\rceil$.

Ainsi, les groupes d'instances activées simultanément sont des motifs tels que définis au chapitre V. Nous pouvons donc représenter les traces symboliques des exemples précédents de la façon suivante :

1. Trace symbolique : $\{(0 \mid [4,2,1,1]), (8 \mid [0,0,0,1]), \ldots\}$;
2. Trace symbolique : $\{(0 \mid [0,0,0,1]), (1 \mid [3,0,0,0]), (2 \mid [1,2,0,0]), (3 \mid [0,0,1,0]), (8 \mid [0,0,0,1]), \ldots\}$.

L'intérêt des motifs et de la trace symbolique est évident sur le premier exemple, un peu moins sur le second. Le scénario le plus défavorable, c'est-à-dire pour lequel la notation introduite n'apporte aucun bénéfice, correspond à des tâches harmoniques et des *offsets* tous différents.

Dans le cas des réseaux AFDX, les périodes sont des puissances de deux, de 1ms à 128ms. D'autre part, au départ d'un même *end-system*, les applications sont ordonnancées hors-ligne selon un schéma bien défini en cycles majeurs et mineurs. De fait, les *offsets* des trames AFDX ne peuvent être identiques. Mais, au niveau des VL, les choses peuvent être différentes. En effet, d'une part les disparités entre les périodes des applications (tâches périodiques) et celles des VL (apparentés à des tâches sporadiques), et d'autre part la configuration de la messagerie (telle application utilise tel VL pour tel message) mènent rapidement les VL à entrer en compétition pour le gain du réseau. Par ailleurs, les arrivées simultanées au niveau des commutateurs ne sont pas rares dès lors que la charge du réseau augmente.

Une génération « au fil des calculs » de la trace symbolique des activations est possible. Pour cela, il faut trier les tâches par ordre de dates d'activation croissantes, puis par ordre de périodes croissantes et parcourir les différentes instances dans l'ordre de leur apparition.

Soit X le tableau initialisé avec les r_i ; on suppose que l'un au moins des X_i est nul, sinon il est possible de s'y rapporter en retranchant le minimum des r_i à tous. Dans notre exemple, $r_i = 0$ pour toutes les tâches. De même, soit t l'instant courant, initialisé à zéro. Les tâches dont le X_i est nul sont activées à l'instant courant t. Dans notre exemple, toutes les tâches sont activées à l'instant initial. Ensuite, dès qu'un X_i est nul, on lui ajoute la période de la tâche associée : $X_i = T_i$. L'instant d'activation suivant t est déduit en lui ajoutant le minimum des X_i et en le retranchant à l'ensemble du tableau. etc. Le tableau 6.1 montre les premières étapes de l'algorithme sur l'exemple considéré.

Pour mener le calcul des lois de distribution des tâches sur l'hyperpériode, nous ne faisons appel aux calculs du chapitre précédent que lorsque l'instant courant est dans cet intervalle (en ayant pris soin de calculer le *backlog* au fur et à mesure). Nous stoppons dès que t est en dehors de l'intervalle.

Comme nous venons de le présenter, il est possible de faire avancer « le temps » aux différents instants où une activation de tâche au moins à lieu de façon relativement simple et d'y mener les calculs souhaités. L'avantage évident de cette approche « au fur et à mesure » est la faible quantité de mémoire qu'elle requiert en $\mathcal{O}(n)$, le nombre de tâches du système. L'inconvénient, c'est son incapacité à proposer des déplacements aléatoires dans le temps, ni vers l'avenir, ni vers le passé.

Pour cela, il faut garder en mémoire les couples $(t \mid \{\text{tâches activées}\})$ plus d'autres informations éventuellement : *backlog*, minimum et maximum des temps de réponse pour les tâches activées à cet instant par exemple. Cependant, l'impact mémoire peut être important,

instant t	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8
0	0*	0*	0*	0*	0*	0*	0*	0*
	100	200	200	200	60	30	30	8
8	92	192	192	192	52	22	22	0*
	92	192	192	192	52	22	22	8
16	84	184	184	184	44	14	14	0*
	84	184	184	184	44	14	14	8
24	76	176	176	176	36	6	6	0*
	76	176	176	176	36	6	6	8
30	70	170	170	170	30	0*	0*	2
	70	170	170	170	30	30	30	2
32	68	168	168	168	28	28	28	0*
	68	168	168	168	28	28	28	8
40	60	160	160	160	20	20	20	0*
	60	160	160	160	20	20	20	8
48	52	152	152	152	12	12	12	0*
	52	152	152	152	12	12	12	8
56	44	144	144	144	4	4	4	0*
	44	144	144	144	4	4	4	8
60	40	140	140	140	0*	0*	0*	4
	40	140	140	140	60	30	30	4
64	36	136	136	136	56	26	26	0*
	36	136	136	136	56	26	26	8

Tableau 6.1 Exemple de génération d'une trace des activations (* = activation).

au pire en $\mathcal{O}(n \cdot \operatorname{ppcm} T_i)$ (n tâches à garder en mémoire pour tout instant d'activation sur la période d'étude).

Un prototype, on le verra, a été développé sur cette base, pour lequel nous avons choisi de générer la trace symbolique en deux temps. Nous commençons par générer un arbre binaire en parcourant chaque tâche et en les ajoutant à leurs instants d'activation ($r_i + k \cdot T_i$, pour k entier positif) dans l'intervalle $[0, \max r_i + 2 \cdot \operatorname{ppcm} T_i[$. Nous obtenons un arbre dont les feuilles sont les couples cherchés. L'avantage de l'arbre binaire est sa complexité intéressante en $\mathcal{O}(n \cdot \ln(n))$ pour la création de structure ordonnées à partir de données quelconques (comme c'est le cas ici). Ensuite, nous linéarisons cet arbre en une liste ordonnée des informations suivantes : instant, ensemble des tâches activées, *backlog* et temps de réponse minimal et maximal à cet instant (voir la section suivante).

Cette méthode n'est pas la plus intéressante pour les systèmes dont l'hyperpériode est grande (*i.e.* de grand $\operatorname{ppcm} T_i$) à cause d'un grand nombre d'instants à garder en mémoire. Néanmoins et encore une fois, les périodes des VL dans un réseau AFDX sont des puissances de deux, donc leur hyperpériode est petite, égale au maximum des périodes. Nous avons

adapté notre implémentation à notre problématique initiale (bien que les exemples montrés dans ce manuscrit ne possèdent pas cette caractéristique de périodes harmoniques).

Dans les sections suivantes, nous poursuivons avec l'exemple introduit page 54 dans le tableau 5.1 et nous considérons un jeu particulier des *offsets*, r_i nul pour toute les tâches. C'est le cas synchrone qui met en évidence le temps de réponse pire cas pour les tâches dans le cas de systèmes mono-processeur et non-préemptifs. Le pire cas pourrait survenir dans l'intervalle $[0, \max r_i + \operatorname{ppcm} T_i[$ que nous négligeons pour le calcul de la loi de distribution, mais pas pour le calcul du *backlog*, des minima et des maxima des temps de réponse (ou des délais) (voir la section suivante).

6.3 Des calculs simples et utiles

À partir de la trace symbolique des activations, il est possible de calculer différents résultats utiles. Nous donnons des formules simples pour trois résultats particulièrement importants de la théorie des systèmes temps réels : le *backlog* à un instant donné et les temps de réponse minimaux et maximaux pour chacune des tâches du système.

Dans notre contexte de réseaux AFDX, le *backlog* est utilisé pour borner l'utilisation maximale des mémoires tampon d'envoi des *end-systems* et des commutateurs et le temps de réponse maximal d'une tâche est utilisé pour borner supérieurement le délai de traversée pire cas d'un commutateur si le scénario d'*offsets* utilisé dans le calcul correspond au pire cas (ce dernier restant encore à déterminer). Nous procédons en deux étapes pour le calcul du *backlog*. Soit t un instant d'activation de $p(t)$ tâches. Nous ajoutons le *backlog* et la contribution des tâches activées à l'instant d'activation précédent, notons $t-1$. Puis, nous retranchons le temps écoulé entre l'instant courant et le précédent, noté Δt.

Ce pseudo-algorithme peut être exprimé à l'aide de deux suites récurrentes : (b_t) et (B_t). La première tient le compte des sommes des durées d'exécution des tâches activées aux différents instants d'activation et la seconde est la suite des *backlog* à ces instants. Ces deux suites sont définies de la façon suivante (pour rappel, la notation $[\bullet]^+$ représente le maximum entre $\bullet$ et zéro) :

$$b_{t\geq 0} = \sum_{i=1}^{p(t)} C_i, \tag{6.1}$$

$$\text{et } B_{t\geq 0} = \begin{cases} 0 & \text{si } t = 0, \\ [B_{t-1} + b_{t-1} - \Delta t]^+ & \text{sinon.} \end{cases} \tag{6.2}$$

Une fois le *backlog* calculé, celui des temps de réponse minimaux et maximaux des tâches est direct. En effet, le temps de réponse d'une tâche est d'autant plus court que cette tâche est devant les autres tâches activées en même temps qu'elle dans la file. Le temps d'attente le plus court, noté $m_t(i)$ pour la tâche i à l'instant d'activation p, s'obtient donc naturellement en plaçant la tâche à la première place libre dans la file et correspond précisément au *backlog* de cet instant. Le résultat est en fait indépendant de la tâche considérée :

$$m_t(i) = B_t \tag{6.3}$$

De même, le temps de réponse d'une tâche est d'autant plus long que cette tâche est derrière les autres tâches activées en même temps qu'elle dans la file. Le temps d'attente le plus long, noté $M_t(i)$, s'obtient donc naturellement en plaçant la tâche en dernier dans la file, après toutes les autres. Il correspond précisément au *backlog* augmenté de la somme des durées d'exécution des autres tâches activées en même temps. De nouveau, le résultat est indépendant de la tâche considérée dans la mesure où quelque soit l'ordre des tâches dans la file, la dernière finira son exécution toujours au même instant :

$$M_t(i) = B_t + b_t \tag{6.4}$$

La complexité totale est au pire en $\mathcal{O}\left(n \cdot \text{ppcm}(T_i)\right)$: au plus $n-1$ sommes pour le calcul de b_t, trois supplémentaires pour le calcul de B_t et m_t et une dernière pour le calcul de M_t pour un total de $n+3$. Calculs à répéter pour tous les instants d'activation dont le nombre est borné par l'hyperpériode, $\text{ppcm}\, T_i$ (à défaut de mieux comme expliqué à la section précédente).

6.4 Algorithme général

Soit t l'instant courant et $J_{i,j}$ l'instance j de la tâche i activée à cet instant. Nous calculons sa loi de distribution des temps de réponse comme expliqué au chapitre précédent. Nous ajoutons à tous les temps de réponse ainsi calculés la valeur du *backlog* à t. Nous répétons ces opérations pour toutes les tâches, sur toute la durée de l'hyperpériode.

Ensuite, nous consolidons les contributions de toutes les instances d'une tâche sur l'ensemble de l'hyperpériode. Pour cela, nous calculons simplement la moyenne arithmétique des distributions (la médiane serait sûrement un meilleur choix pour être moins sensible aux valeurs extrêmes, voir chapitre VII). C'est-à-dire que pour chaque valeur entre le minimum et le maximum des temps de réponse des distributions de toutes les instances de la tâche considérée, nous ajoutons les poids de toutes les distributions et divisons cette somme par le nombre d'instants d'activation de la tâche sur l'intervalle (égal à $\text{ppcm}\, T_i \,/\, T_j$ par construction).

Cela suppose que chaque loi de distribution, ou de manière équivalente, que chaque instant d'activation a autant d'importance que les autres. Ceci est vrai pour la raison suivante. Considérons que le système évolue depuis quelque temps et prélevons une instance à un instant quelconque. Cet instant, ramené dans l'intervalle de l'hyperpériode par congruence (modulo l'hyperpériode, égale au $\text{ppcm}\, T_i$), peut correspondre à n'importe lequel des différents instants d'activation de sa tâche dans l'intervalle. De cette façon, on pondère la probabilité de chaque valeur, dans chaque distribution, par le poids équiprobable de l'instance ($\frac{\text{ppcm}\, T_i}{T_j}$ pour la tâche j). En termes plus mathématiques, nous exploitons le principe de partitionnement de l'univers (ensemble de toutes les possibilités) :

$$\mathcal{P}(\text{délai} = x) = \sum_{t \in \text{activations}} \mathcal{P}(\text{activation} = t) \cdot \mathcal{P}(\text{délai} = x \mid \text{activation} = t) \tag{6.5}$$

Ainsi, la loi de distribution d'une tâche est égale à la moyenne arithmétique des lois de distribution de ses instances. Dit autrement, la probabilité d'un temps de réponse particulier d'une tâche temps réel dans un système est égale à la moyenne des probabilités que les instances de cette tâche subissent ce temps de réponse.

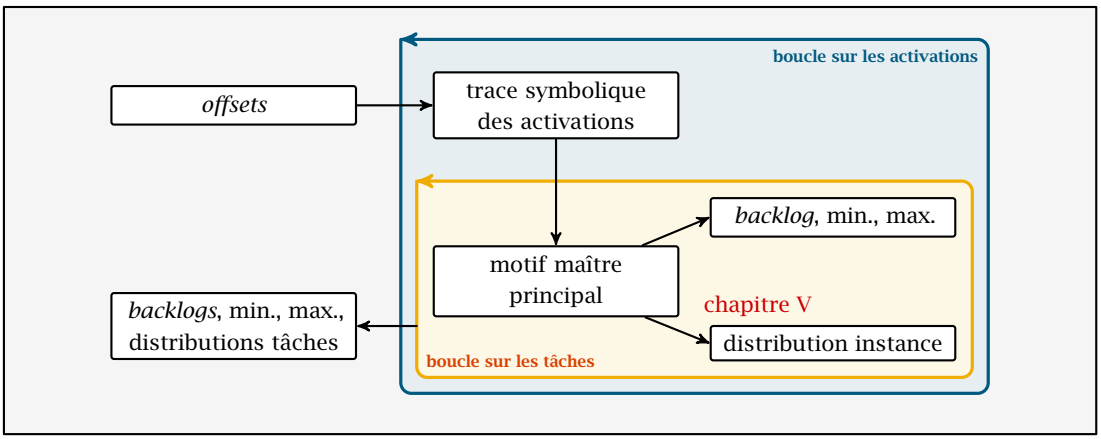

Figure 6.1 Diagramme récapitulatif du calcul des lois de distribution des temps de réponse des tâches d'un système temps réel.

6.5 Exemple d'illustration

Le système étudié est composé de huit tâches. Par commodité, nous rappelons ici le tableau 5.1 de la page 54. Considérons la tâche τ_1 et calculons la loi de distribution de son temps de réponse ainsi que son temps de réponse pire cas — qui peut être différent du maximum de la distribution si le jeux d'*offsets* ne conduit pas à ce supremum ou qu'il intervient avant la partie cyclique (voir les sections précédentes).

tâche	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8
C_i	10	10	10	10	4	4	5	1
T_i	100	200	200	200	60	30	30	8

Nous considérons le cas synchrone, toutes les tâches sont donc activées en même temps à l'instant initial $t = 0$. En particulier, la tâche 1 est activée à l'instant initial, puis tous les multiples de 100 jusqu'à 1200 exclu, où nous arrêtons nos calculs pour le calcul de la loi de distribution des temps de réponse de la tâche 1 (et pour toutes les autres aussi d'ailleurs).

À l'instant initial, les huits tâches sont activées. Le motif maître principal est $[4, 2, 1, 1]$ et le motif maître secondaire associé à τ_1 est donc $[3, 2, 1, 1]$. Ce motif maître secondaire correspond à $(3+1)(2+1)(1+1)(1+1)$ motifs esclaves, soit 48 permutations. L'exemple à la fin du chapitre précédent est identique, sauf que nous y considérions J_4, une instance de la tâche 4. Comme τ_1 et τ_4 appartiennent toutes deux à la classe d'équivalence $\mathbb{C}_1^{10}$, et dans la mesure où à l'instant initial le *backlog* est nul (comme l'hypothèse du chapitre précédent), la loi de distribution figure 5.3 est valide aussi pour $J_{1,1}$, l'instance de τ_1 à l'instant initial et pour tout instant où toutes les tâches sont actives et pour lequel le *backlog* est nul. En l'occurrence, le premier instant à considérer pour le calcul de la loi de distribution est $\max r_i + \operatorname{ppcm} T_i = 600$, le *backlog* y est nul et toutes les tâches y sont activées. En d'autres termes, cette loi de distribution est aussi valide pour $J_{1,7}$ à cet instant. C'est aussi le seul instant répondant à ces critères sur l'intervalle étudié : l'hyperpériode.

Nous avons notre premier contributeur à la distribution globale ! Il faut normaliser cette distribution en divisant les valeurs par le nombre d'instances de τ_1 activée sur l'hyperpériode

délai	Probabilité (~)						délai	Probabilité (~)					
	τ_1	$\tau_{2\text{ à }4}$	τ_5	τ_6	τ_7	τ_8		τ_1	$\tau_{2\text{ à }4}$	τ_5	τ_6	τ_7	τ_8
0	0.4625	0.1750	0.1791	0.2145	0.2145	0.3103	26	0.0017	0.0035	0.0021	0.0010		0.0133
1	0.0196	0.0392	0.0267	0.0633	0.0633	0.0266	27			0.0166	0.0333	0.0416	0.0166
2						0.0266	28	0.0017	0.0035	0.0166	0.0083	0.0010	0.0136
3						0.0400	29	0.0065	0.0130	0.0021	0.0010	0.0017	0.0005
4	0.0337	0.0119	0.0684	0.0342	0.1934	0.0071	30	0.0232	0.0464	0.0050	0.0025	0.0007	0.0055
5	0.0188	0.0099	0.0940	0.1720	0.0755	0.0302	31	0.0672	0.1345	0.0014	0.0007	0.0423	
6	0.0009	0.0019	0.0255	0.0627		0.0933	32			0.0333	0.0666	0.0250	0.0266
7						0.0266	33	0.0029	0.0059				0.0271
8	0.0148	0.0019			0.0502	0.0300	34	0.0101	0.0202	0.0014	0.0007	0.0014	0.0137
9	0.0303	0.0051	0.1005	0.0502	0.0376	0.0334	35	0.0025	0.0051	0.0038	0.0019	0.0023	0.0135
10	0.0601	0.1202	0.1241	0.0870	0.0494	0.0062	36	0.0009	0.0019	0.0023	0.0011	0.0250	
11	0.0363	0.0726	0.0023	0.0011	0.0011	0.0133	37				0.0250	0.0011	
12						0.0266	38	0.0009	0.0019				0.0003
13	0.0422	0.0011				0.0500	39	0.0049	0.0099	0.0023	0.0011	0.0035	0.0139
14	0.0065	0.0130	0.0523	0.0261	0.0648	0.0039	40	0.0059	0.0119	0.0075	0.0037	0.0001	0.0053
15	0.0065	0.0130	0.0371	0.0435	0.0055	0.0003	41			0.0005	0.0002	0.0002	
16	0.0017	0.0035	0.0014	0.0007			43	0.0029	0.0059				0.0009
18	0.0017	0.0035	0.0083	0.0041	0.0382	0.0157	44	0.0208	0.0416	0.0005	0.0002	0.0005	0.0001
19	0.0053	0.0107	0.0764	0.0382	0.0090	0.0281	45			0.0023	0.0011	0.0017	0.0000
20	0.0398	0.0797	0.0133	0.0066	0.0017	0.0324	46			0.0017	0.0008		0.0133
21	0.0517	0.1035	0.0021	0.0010	0.0010	0.0133	48					0.0008	0.0002
22					0.0041	0.0011	49			0.0017	0.0008	0.0062	0.0004
23	0.0017	0.0035	0.0416	0.0208	0.0291	0.0024	50			0.0125	0.0062		
24	0.0065	0.0130	0.0271	0.0135	0.0021	0.0005	53						0.0016
25	0.0053	0.0107	0.0042	0.0021	0.0021	0.0136							

Tableau 6.3 Lois de distribution des délais de toutes les tâches.

(ici, l'intervalle $[600, 1200[$) : il y en a six. Il se trouve qu'à chacun de ces instants, le *backlog* est nul (ce n'est pas le cas à l'instant 420 par exemple où il est de 23). En fait, le *backlog* est nul pour environ un tiers des instants d'activation sur l'hyperpériode et pour ce jeu d'*offsets* particulier (tous nuls).

D'autre part, les instances $J_{1,8}$ et $J_{1,12}$ sont activées seules. De plus, comme le *backlog* est le même aux deux instants, les lois de distribution sont les mêmes également. Aux instants 800 et 1000, les instances $J_{1,9}$ et $J_{1,11}$ sont activées avec les mêmes autres tâches : τ_2, τ_3, τ_4 et τ_8. Le motif maître principal correspondant est alors $[4, 0, 0, 1]$; il y a 120 permutations possibles de ces cinq instances dans la file. Grâce au chapitre précédent, nous nous ramenons à l'étude de huit scénarios différents seulement. L'instance $J_{1,10}$ est elle activée avec d'autres tâches : τ_5, τ_6 et τ_7. Six scénarios représentent 24 permutations possibles des quatres tâches.

Les résultats de chacune des instances de τ_1 sur l'hyperpériode ($[600, 1200[$) et de τ_1 elle-même sont regroupés dans le tableau 6.2. Les distributions des tâches 2 à 8 sont reportées dans le tableau 6.3, mais sans le détails des calculs intermédiaires. Tous ces résultats ont été obtenus à l'aide d'un prototype que nous avons développé en Fortran 90 aux cours de ces travaux. Voir l'annexe B pour plus d'information sur le prototype.

délai	(temps de réponse)	probabilités (~)						
		$J_{1,7}$	$J_{1,8}$	$J_{1,9}$	$J_{1,10}$	$J_{1,11}$	$J_{1,12}$	τ_1
0	10	5040/40320	1	1/5	1/4	1/5	1	0.4625
1	11	720/40320		1/20		1/20		0.0196
4	14	1440/40320			1/6			0.0337
5	15	1200/40320			1/12			0.0188
6	16	240/40320						0.0009
8	18	240/40320			1/12			0.0148
9	19	624/40320			1/6			0.0303
10	20	2448/40320		3/20		3/20		0.0601
11	21	720/40320		1/10		1/10		0.0363
13	23	144/40320			1/4			0.0422
14	24	1584/40320						0.0065
15	25	1584/40320						0.0065
16	26	432/40320						0.0017
18	28	432/40320						0.0017
19	29	1296/40320						0.0053
20	30	1584/40320		1/10		1/10		0.0398
21	31	432/40320		3/20		3/20		0.0517
23	33	432/40320						0.0017
24	34	1584/40320						0.0065
25	35	1296/40320						0.0053
26	36	432/40320						0.0017
28	38	432/40320						0.0017
29	39	1584/40320						0.0065
30	40	1584/40320		1/20		1/20		0.0232
31	41	144/40320		1/5		1/5		0.0672
33	43	720/40320						0.0029
34	44	2448/40320						0.0101
35	45	624/40320						0.0025
36	46	240/40320						0.0009
38	48	240/40320						0.0009
39	49	1200/40320						0.0049
40	50	1440/40320						0.0059
43	53	720/40320						0.0029
44	54	5040/40320						0.0208

Tableau 6.2 Lois de distribution des délais et temps de réponse de τ_1.

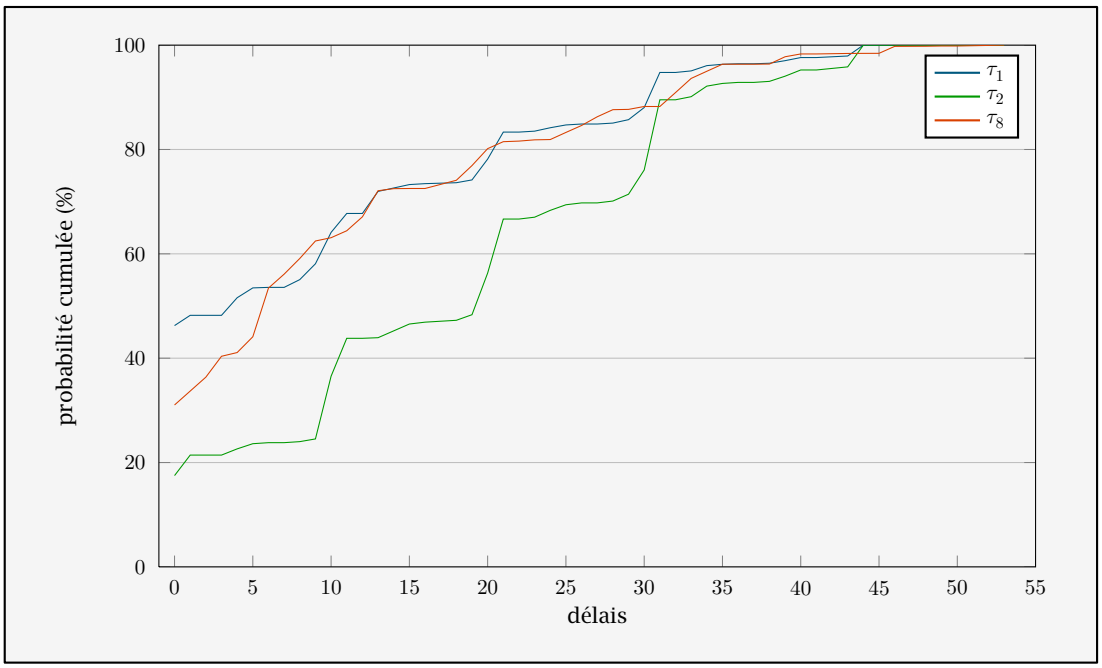

Figure 6.2 Fonctions de répartition de τ_1, τ_2 et τ_8 par rapport aux délais (cas synchrone).

6.6 Interprétation des résultats

Les lois de distribution pour les différentes tâches sont présentées en fonction des délais rencontrés par les tâches plutôt qu'en fonction de leurs temps de réponse afin de simplifier les comparaisons entre les tâches. Pour obtenir les temps de réponse, il suffit d'ajouter la durée d'exécution. Enfin, et toujours en vue de favoriser les comparaisons, nous traçons plutôt les lois de distribution cumulées (ou fonctions de répartition). Ce sont les fonctions qui à un délai donné x associent la probabilité pour une tâche de subir un délai inférieur ou égal à x, alors que les lois de distribution y associent la probabilité pour cette tâche de subir exactement ce délai. D'une manière générale la probabilité qu'une tâche subisse un délai donné ou moins est plus intéressant que la probabilité de subir un délai isolé. Les deux représentations contiennent évidemment la même information, mais la lecture directe est plus aisée dans le cas de la fonction de répartition.

Nous avons tracé à la figure 6.2 les fonctions de répartition des tâches 1, 2 et 8 en fonction des délais subis par les tâches, en bleu, vert et rouge respectivement. Ces lois de distribution ne sont valables que pour le jeu d'*offsets* synchrone. À titre de comparaison, nous avons tracé les mêmes courbes pour un jeu d'*offsets* quelconque à la figure 6.3.

Première remarque, nous constatons que τ_1 ne subit aucun délai presque la moitié du temps dans cette configuration d'*offsets* synchrones. Ce résultat n'est pas surprenant si l'on analyse les raisons d'un délai nul : il faut que la tâche soit placée la première dans la file d'attente et que le *backlog* soit nul. Le *backlog* est bien nul en tous les instants d'activation de la tâche et en tous ces instants, τ_1 peut être la première. À chaque instant, τ_1 peut ne subir aucun délai et la combinatoire est favorable avec une probabilité de $1 / p$ avec p le nombre de tâches activées à l'instant considéré (le poids du motif est $(p-1)!$ et le nombre total des permutations est $p!$). Cela signifie que cette probabilité est égale à 1 quand τ_1 est activée seule

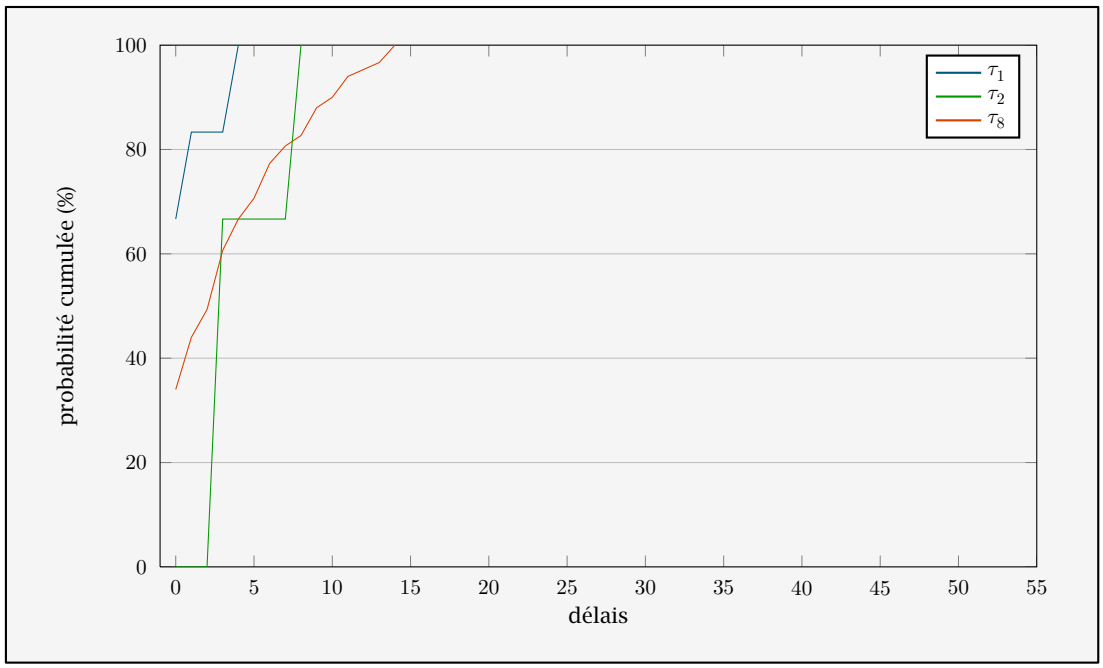

Figure 6.3 Fonctions de répartition de τ_1, τ_2 et τ_8 par rapport aux délais (cas asynchrone).

(instances $J_{1,8}$ et $J_{1,12}$) ! Ce résultat est d'autant plus vrai que les *offsets* sont « bien répartis » entre eux comme nous pouvons le constater sur la figure 6.3 (les *offsets* des tâches 1 à 8 ont été tirés aléatoirement et sont respectivement 66, 179, 11, 139, 11, 26, 1 et 7). Cela revient à lisser la charge plutôt que de tout concentrer en certains instants comme les instants zéro (toutes les tâches activées), 200 (τ_1 à τ_4 et τ_8), 300 (τ_1 et τ_5 à τ_7), 400, 600, 800, 900, 1000, 1200, etc.

Deuxième remarque, nous constatons certaines valeurs privilégiées des délais subis par la tâche 1 dans le tableau 6.2. Ces valeurs privilégiées sont regroupées par paquets : 1, 4 et 5, 8 et 9, 10 et 11, 13, 20 et 21, 30 et 31 et enfin 44. Ces valeurs sont obtenues à plusieurs instants et/ou avec des poids ou combinatoires importants. Les délais 1, 10 et 11, 20 et 21 et 30 et 31 sont possibles pour les instances $J_{1,7}$, $J_{1,9}$ et $J_{1,11}$ et les délais 4 et 5, 8 et 9 et 13 sont possibles pour les instances $J_{1,7}$ et $J_{1,10}$ et ont une combinatoire importante pour l'instance $J_{1,7}$. Le délai 44 n'est obtenu qu'avec l'instance $J_{1,7}$, mais sa combinatoire est élevée et pèse plus que les multiples possibilités d'occurrence du délai 1 par exemple.

Troisième remarque, pour le cas synchrone, les tâches 2 à 4 sont identiques (mêmes durées d'exécution, mêmes périodes et mêmes *offsets*), donc les résultats sont également strictement identiques. Pour ces tâches-là, la probabilité pour une instance quelconque de subir un délai nul est plus faible et plusieurs valeurs sont d'égales importances (environ) au délai nul. Ce phénomène s'explique justement parce que les tâches sont identiques : elles sont donc toujours activées toutes les trois en même temps, plus une fois sur deux avec τ_1. Les valeurs « pics » (voir les sauts sur la figure 6.2) sont obtenues pour des délais nul, 10, 20 et 30, les différentes combinaisons des durées d'exécution des tâches 1 à 4. Les pics sont moins élevés en ces valeurs pour τ_1 car elle n'est activée qu'une fois sur deux avec les tâches 2 à 4.

Quatrième remarque, le *backlog* étant nul à l'instant $\max r_i + \operatorname{ppcm} T_i = 600$, la partie

cyclique de l'ordonnancement a en fait débuté à l'instant zéro. Cela ne serait pas le cas si le *backlog* n'avait pas été nul, les conditions auraient alors été différentes : mêmes tâches activées, mais *backlogs* différents et donc, lois de distribution différentes. Cela implique notamment que le temps de réponse maximal calculé dans le tableau 6.2 est le vrai maximum pour cette configuration des *offsets*. D'autre part, puisque sous les conditions de système mono–processeur à ordonnanceur non–préemptif nous savons que cette configuration particulière des *offsets* conduit à faire apparaître le temps de réponse pire cas de chacune des tâches, nous en déduisons que le temps de réponse pire cas de τ_1 dans ce système est 54.

Cinquième remarque, le phénomène de lissage est flagrant si nous comparons les figures 6.2 et 6.3. Les tâches ne subissent que peu de délais différents, et tous relativement faibles comparé au maximum de 54.

6.7 Conclusions

Dans ce chapitre, nous avons mis en évidence les problèmes d'explosion combinatoire liés aux multiples combinaisons des instances actives dans la file d'attente d'un processeur d'un système temps réel, avec un seul niveau de priorité, en contexte non–préemptif.

Pour résoudre ce problème, nous avons introduit une notation symbolique et efficace des traces des activations des tâches : les instances y sont regroupées par date d'activation commune. Nous avons utilisé cette trace symbolique pour calculer différentes propriétés temporelles intéressantes : le *backlog* du sytème aux instants d'activation et les temps de réponse minimaux et maximaux des tâches.

Nous avons ensuite utilisé cette trace symbolique et les résultats du chapitre précédent pour présenter notre algorithme original et efficace du calcul la loi de distribution des temps de réponse (ou des délais) de tâches temps réel. Ces derniers ne sont plus calculés pour chaque permutations, mais pour des motifs qui en représentent plusieurs. Ces derniers sont ensuite pondérés et normalisés. Cet algorithme est valable pour des ensembles de tâches strictement périodiques à *offsets* connus dans le cadre de systèmes temps réel à un seul niveau de priorité.

Que dire lorsque les *offsets* sont différents à chaque démarrage ? De deux choses l'une, soit la séquence de démarrage d'un système temps réel est connue à l'avance, soit elle ne l'est pas. Dans ce cas, il n'est pas impossible que les *offsets* des tâches soient différents d'un allumage du système à un autre.

D'autre part, soit les tâches d'un système temps réel distribué comme un réseau sont synchronisées sur une horloge commune ou partagée, soit leurs horloges propres dérivent les unes par rapport aux autres. En effet, les quartz de ces horloges ne sont pas strictement identiques et leur fréquences sont légèrement différentes. Cela suffit à ce que les horloges dérivent dans le temps.

Bien que différent, la conséquence de ces deux phénomènes est identique : les lois de distribution des temps de réponse calculées pour un seul jeu d'*offsets* sont insuffisantes pour caractériser le comportement des systèmes temps réel dans la durée. Il faut mener ces calculs pour plusieurs jeux d'*offsets*, et même, *beaucoup* de jeux.

Dans la pratique, il y a bien trop de combinaisons des *offsets* pour effectuer un calcul complet. Pour notre exemple, il existe $5.76 \cdot 10^{11}$ jeux d'*offsets* différents. Exprimé en secondes, cela représente plus de 18264 années ; en bits, cela représente 576 Go ! Dans les faits, il faut adopter des approches statistiques ou réduire le périmètre des *offsets* en considérant des hypothèses supplémentaires, comme le fait que les tâches démarrent toutes en moins de X et alors restreindre les possibilités à des quantités plus acceptables. Nous présentons dans le chapitre suivant une méthode statistique de type Monte Carlo afin d'analyser l'impact d'*offsets* différents.

Remarque Les résultats de ce chapitre et du précédent ont fait l'objet d'un dépôt de brevet en France et en Europe : « Procédé de calcul de la distribution de la durée de traversée d'un commutateur de communication multiplexé », numéro de dépôt 11 01935 [50]. Ce fût une expérience longue, parfois douloureuse, mais finalement enrichissante et ce à différents niveaux : en terme de rédaction et de mise à niveau du discours d'abord, d'un point de vue juridique ensuite, la législation entourant les brevets étant assez complexe.

Étude statistique de réseaux AFDX

VII

Noud avons calculé les lois de distribution exactes d'instances et de tâches temps réel (pour un seul jeu d'*offset*), nous évaluons dans ce chapitre une approximation de la loi de distribution de tâches réel dans un contexte mono–processeur, non–préemptif, à un seul niveau de priorité et pour tout jeu d'*offsets*.

Pour cela, nous montrons que dans le cas général, même si le nombre de jeux d'*offsets* (fondamentalement) différents entre les tâches d'un système temps réel peut se ramener à un nombre fini, il reste trop important pour envisager une résolution directe du problème. Nous rappelons ensuite les grands résultats des méthodes de Monte Carlo et montrons que ce sont des méthodes statistiques adaptées à notre problématique. Enfin, nous développons un exemple et discutons de la pertinence de nos résultats en les comparant aux délais pire cas qui sont des suprema des lois de distribution des tâches temps réel. Cette approche est équivalente à une approche par simulation.

7.1 Une combinatoire imposante et incompressible

Combien de jeux différents d'*offsets* existe-il au juste ? Naïvement, il y en a une infinité. Mais a y regarder de plus près, ce n'est pas tout à fait vrai, du moins pas si l'on se place « loin » de la dernière « première activation » vers l'avenir. Après une montée en charge du système, ce dernier atteint un état stationnaire.

Si la charge processeur U est strictement inférieure à 100 % à l'instant de la dernière activation, alors après un certain nombre d'hyperpériodes, le *backlog* sera « absorbé » dans les trous du processeur (la marge pour arriver à 100 %). Dès que le *backlog* est nul à un instant d'activation, le passé n'a plus d'importance. Dit autrement, chaque instant d'activation où le *backlog* est nul peut être le minimum des *offsets* pour un jeu différent.

Au–delà d'une certaine durée, les *offsets* initiaux n'ont plus d'importance. En mécanique du solide, cela s'appelle le principe de Saint Venant : loin des causes, les conséquences sont négligeables. De fait, on peut toujours se ramener à un instant d'activation de *backlog* nul et considérer qu'il s'agit de l'instant initial puisque le passé n'y a plus d'impact sur l'avenir.

De là, nous pouvons considérer que les prochaines activations des différentes tâches sont leurs instants de première activation. Se faisant, nous constatons que la première activation de chaque tâche intervient alors dans un intervalle de durée au plus égale à la période de la tâche, c'est-à-dire que pour tout i

$$0 \leq r_i < T_i(\text{émissioninstantanéedelatâche}). \tag{7.1}$$

Grâce à ce raisonnement simple, nous venons de limiter le nombre de jeux d'*offsets* différents, en tous cas si l'on se place suffisamment loin de l'instant initial du système pour s'éloigner des phénomènes transitoires et se placer dans un état stationnaire. À partir de maintenant, nous ferons cette hypothèse systématiquement. En fait, ce résultat a été démontré formellement par Goosens et Cucu-Grosjean dans [28]. Ils démontrent que du point de vue de l'évaluation de bornes pire cas, le nombre de jeux d'*offsets* différents à prendre en compte est égal au produit des périodes des tâches du système. Ils ont même fait mieux ; ils ont montré que ce nombre pouvait encore être divisé par l'hyperpériode, réduisant d'autant la combinatoire à considérer.

C'est un résultat que nous avons (re)trouvé indépendamment par induction, mais sans apporter de démonstration. Nous n'avons d'ailleurs pas remarqué tout de suite que les résultats étaient identiques. En effet, si l'on décompose T_i en facteurs premiers et que l'on note $k_{i,p}$ la puissance du facteur premier p dans la décomposition (la suite $(k_{i,p})_p$ étant de support fini), nous sommes arrivés à la conclusion tout à fait équivalente que le nombre des *offsets* différents à considérer est égal à

$$\frac{\prod_i T_i}{\prod\limits_{p \text{ premier}} p^{\max_i\{k_{i,p}\}}} = \frac{\prod_i T_i}{\operatorname{ppcm} T_i}. \tag{7.2}$$

Cependant, même fini et diminué, ce nombre reste très grand. Cela est d'autant plus vrai si les tâches sont multipériodiques et que l'hyperpériode est égale au maximum des périodes. Nous rappelons que c'est le cas des réseaux AFDX. À titre d'exemple, pour l'exemple fil rouge proposé, le nombre d'*offsets* différents s'élèvent à $5.76 \cdot 10^{11}$. Exprimé en secondes, cela représente environ 18265 années.

La solution pour minimiser ce rapport est de maximiser le ppcm en dessous, c'est-à-dire l'hyperpériode. Mais alors, la longueur de l'intervalle d'étude, égale à l'hyperpériode, est elle-même maximale. Nous sommes dans une impasse. D'un côté, nous réduisons l'intervalle d'étude, mais il faut multiplier les *offsets*, de l'autre, nous diminuons les jeux d'*offsets* différents en alongeant la fenêtre d'étude. Que faire ? Nous proposons d'utiliser une méthode statistique : nous estimons les propriétés d'intérêt sur des échantillons représentatifs de la population plutôt que sur la population elle-même. Dans les sections suivantes, nous présentons les méthodes de type Monte Carlo que nous avons utilisées pour évaluer les lois de distribution des délais maximaux et minimaux des tâches d'un système temps réel à *offsets* inconnus.

7.2 Une approche statistique adaptée

Une méthode statistique consiste généralement à faire correspondre des modèles analytiques paramétrés à des lois de distribution réelles à l'aide d'échantillons tirés aléatoirement

parmi une vaste population. Ces échantillons servent alors à estimer les paramètres des modèles analytiques. Par exemple, supposons que la répartition des individus d'un pays comme la France selon leur poids suit une loi normale, c'est-à-dire que la courbe de la distribution est une Gaussienne d'équation

$$f(x;\mu,\sigma) = \mathcal{N}(\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} \cdot \exp\left(-\frac{1}{2} \cdot \left(\frac{x-\mu}{\sigma}\right)^2\right). \tag{7.3}$$

Différentes méthodes permettent alors d'estimer la valeur des paramètres μ et . La connaissance d'un modèle analytique de la loi de distribution du poids des français permet ensuite d'estimer le poids moyen de la population, le pourcentage de personnes en surpoids, ou encore l'intervalle où se situe telle ou telle proportion de la population, etc.

Comme toute méthode statistique, les méthodes de Monte Carlo consistent à estimer certaines propriétés. Elles consistent à effectuer des tirages aléatoires d'échantillons parmi une population d'événements possibles et à leur appliquer une fonction pour obtenir d'autres échantillons et en déduire d'autres propriétés sur la population totale. En effet, il n'est pas toujours possible de procéder à des tirages donnant directement accès à la propriété recherchée ou bien il n'en existe pas de modèle analytique *a priori*. Concrètement, étant donnée la loi de distribution d'une variable aléatoire X, nous cherchons à déterminer la loi de distribution d'une variable aléatoire $Y = f(X)$ pour une fonction f quelconque. Par exemple, la répartition de l'indice de masse corporel de la population française, ou dans le cas qui nous préoccupe, la loi de distribution, en fonction des *offsets*, des lois de distribution des délais des tâches d'un système temps réel.

Dans un souci de simplification, nous ne nous intéressons dans ce manuscrit qu'aux lois de distribution des délais, tous jeux d'*offsets* confondus, ainsi qu'aux lois de distribution des délais maximaux et minimaux de ces tâches et non aux « lois de distribution des lois de distribution ». Par ailleurs, nous estimons que la loi de distribution (par rapport aux *offsets*) des lois de distribution (par rapport aux délais) des tâches présente moins d'intérêt compte tenu de la dimension supplémentaire et du faible gain en terme de compréhension du système.

Les méthodes de type Monte Carlo présentent plusieurs intérêts. (1) Elles sont simples et intuitives (en fait, nous les utilisons plus ou moins naturellement tous les jours). (2) Elles permettent d'estimer différentes propriétés sur des populations parfois trop importantes pour traiter tous leurs individus, ou de manière peu commode (les populations à l'échelle de pays par exemple). (3) Elles peuvent servir de première étape dans l'élaboration d'un modèle analytique en estimant les lois de distribution des échantillons. (4) Les résultats produits sont représentatifs à la hauteur des échantillons. Cette propriété est particulièrement intéressante si l'on ne s'intéresse qu'à un sous-ensemble d'une population. Par exemple, le sous-ensemble des *offsets* tels que tous les *offsets* sont dans un intervalle borné (parce que l'on sait que la phase d'initialisation est bornée par exemple), ou tels que certains *offsets* sont toujours plus grands que d'autres parce qu'un ordre d'activation est connu, etc.

D'un autre côté, ces méthodes s'apparentent toujours à de la simulation (le système simulé étant modélisé par notre algorithme, les entrées du système étant les *offsets* et les sorties les lois de distribution des délais). En particulier, les échantillons ne sont généralement

qu'une partie infime de la population. Il faut alors être prudent quant aux conclusions. Il faut prendre soin par exemple de ne pas parler de LA moyenne d'une population sans donner un intervalle ou une valeur de confiance.

Après un rappel des principaux résultats des méthodes de Monte Carlo, et jusqu'à la fin de ce chapitre, nous nous intéressons aux échantillons des jeux d'*offsets* des tâches d'un système temps réel. Nous les utilisons en entrée de notre algorithme présenté au chapitre VI pour calculer les différentes lois de distribution des délais : loi générale, loi des maxima et loi des minima des tâches du système.

7.3 Rappels importants de la théorie des statistiques

Par définition, une variable aléatoire réelle est une variable aléatoire à valeurs réelles. Par la suite, nous désignons par $(Y_i)_i$ une suite de variables aléatoires réelles, indépendantes et de même loi qu'une variable Y (Y est une simple notation pour décrire commodément la loi commune des Y_i).

Pour tout entier n positif, nous posons $S_n = \sum Y_i$ et nous utilisons les notations classiques $\mathrm{E}[Y]$, σ_Y^2 et σ_Y pour désigner respectivement l'espérance, la variance et l'écart type de la variable Y. L'espérance d'une variable aléatoire réelle est sa valeur *moyenne* et « positionne la distribution » ; la variance et l'écart type rendent compte de la dispersion des valeurs prises par la variable autour de son espérance. Commençons par rappeler deux résultats fondamentaux du domaine des statistiques : la loi des grands nombres et le théorème limite central. Pour les détails, le lecteur intéressé pourra se rapporter à [7] par exemple.

– **Loi (forte) des grands nombres**

  Si $\mathrm{E}[|Y|]$ est finie, alors la limite de la suite $(S_n / n)_n$ tend presque sûrement vers $\mathrm{E}[Y]$ quand n tend vers l'infini. Soit $(y_i)_{i\leq n}$ une réalisation des Y_i, c'est-à-dire un tirage aléatoire de taille n de la variable Y. La suite $(S_n / n)_n$ n'est rien d'autre que la moyenne arithmétique des y_i que nous notons $\overline{y_n}$. On dit que la moyenne arithmétique $\overline{y_n}$ de l'échantillon $(y_i)_{i\leq n}$ est un estimateur sans biais de l'espérance de Y.

$$\mathrm{E}[|Y|] < \infty \Longrightarrow \overline{y_n} \xrightarrow[n\to\infty]{} \mathrm{E}[Y]\,. \tag{7.4}$$

  Ce résultat est conforme à l'intuition (ce qui est rassurant). Pour estimer la valeur *moyenne* d'une variable aléatoire réelle, il faut tirer un échantillon de grande taille et calculer sa moyenne arithmétique.

– **Théorème limite central**

  Si $\mathrm{E}\left[Y^2\right]$ est finie, alors la suite $(\overline{y_n})_n$ converge en loi vers une variable aléatoire de loi normale (gaussienne) $\mathcal{N}(\mathrm{E}[Y], \sigma_Y^2 / n)$.

$$\mathrm{E}\left[Y^2\right] < \infty \Longrightarrow \overline{y_n} \xrightarrow[n\to\infty]{} \mathcal{N}(\mathrm{E}[Y], \sigma_Y^2 / n). \tag{7.5}$$

  De nouveau, ce résultat est très intuitif. Plus l'échantillon est grand, plus la moyenne arithmétique est proche de la valeur de l'espérance véritable de Y. Ce résultat dit aussi que la convergence de la moyenne vers l'espérance est en $1 / \sqrt{n}$ (car l'écart type qui

caractérise la dispersion autour de la moyenne est égale à la racine de la variance), c'est-à-dire que pour gagner un ordre de grandeur en précision, il faut un échantillon cent fois plus grand. Enfin, un mot sur le mode de convergence « en loi ». Il s'agit du mode de convergence le plus faible. Dire qu'une suite de variable aléatoire $(Y_i)_i$ tend en loi vers une variable Y, que l'on note $Y_i \xrightarrow{\mathcal{L}} Y$, signifie que la loi de distribution des Y_i tend vers celle de Y. Autrement dit, quelque soit le réel r, alors $\mathcal{P}(Y_n = r) \xrightarrow[n\to\infty]{} \mathcal{P}(Y = r)$.

Considérons X, une variable aléatoire prenant ses valeurs dans un ensemble dénombrable Ω et f une fonction de Ω dans un sous-ensemble borné de $\mathbb{N}$. Nous souhaitons estimer la loi de distribution de la variable aléatoire $Y = f(X)$ à valeurs entières. Plus exactement, nous nous intéressons à la loi de distribution de la variable aléatoire Y, c'est-à-dire à la fonction P^Y qui à tout entier y associe la probabilité que $Y = f(X)$ soit égale à y : $P^Y(y) = \mathcal{P}(f(X) = y)$. Soit $(X_i)_i$ une suite de variables qui convergent en loi vers X, alors la suite $(Y_i = f(X_i))_i$ converge en loi vers Y.

$$X_i \xrightarrow{\mathcal{L}} X \quad \text{et} \quad f : \Omega \to \mathbb{N} \Longrightarrow f(X_n) \xrightarrow{\mathcal{L}} f(X) \qquad \text{soit} \qquad Y_n \xrightarrow{\mathcal{L}} Y. \tag{7.6}$$

Ce résultat est intéressant car en prenant un échantillon des Y_i assez grand, nous pouvons approcher la loi de distribution de la population Y et en déduire un certain nombre de propriétés sur la population. Dans ces travaux, nous ne nous intéressons qu'à caractériser la forme de la loi de distribution, sans essayer d'approcher la loi elle-même par une optimisation paramétrée. En particulier, nous nous intéressons à l'espérance, à la variance, à la médiane et aux coefficients d'aplatissement et de dissymétrie de la variable Y. Les nouvelles notions sont introduites dans les paragraphes suivants.

Pour estimer $\mathrm{E}[Y]$ et σ_Y^2, nous devons utiliser des estimateurs sur les échantillons. En l'occurrence, nous en avons déjà un pour l'espérance grâce au théorème limite central. Il suffit de vérifier les hypothèses sur Y : l'espérance et la variance de Y sont nécessairement finies car Y prend ses valeurs sur un ensemble borné, donc le théorème limite central est applicable. Cela signifie que la moyenne arithmétique $\overline{y_n}$ est un estimateur sans biais de $\mathrm{E}[Y]$ qui suit une loi normale d'espérance $\mathrm{E}[Y]$ et de variance σ_Y^2 / n :

$$\overline{y_n} \xrightarrow[n\to\infty]{} \mathrm{E}[Y] \qquad \text{tel que} \qquad \mathrm{E}[\overline{y_n}] = \mathrm{E}[Y] \quad \text{et} \quad \sigma^2_{\overline{y_n}} = \sigma_Y^2 / n. \tag{7.7}$$

La variance de l'échantillon $s_n^2 = \sum_{i=1}^n (y_i - \overline{y_n})^2 \,/\, (n-1)$ est un estimateur sans biais également de la variance de la population. Tout comme la moyenne arithmétique de l'échantillon, s_n^2 est une variable aléatoire, elle suit une loi (inconnue) d'espérance σ_Y^2 et de variance $\sigma^2_{s_n^2}$:

$$s_n^2 \xrightarrow[n\to\infty]{} \sigma_Y^2 \qquad \text{tel que} \qquad \mathrm{E}\left[s_n^2\right] = \sigma_Y^2 \quad \text{et} \quad \sigma^2_{s_n^2} = \sigma_Y^4 \left(\frac{2}{n-1} + \frac{\gamma_2}{n} \right). \tag{7.8}$$

Le terme γ_2 désigne le *kurtosis* normalisé ou coefficient d'aplatissement normalisé de la distribution des $(y_i)_i$. Plus il est grand, plus la courbe est « piquée » autour de son espérance. À l'inverse, plus il est faible, plus la distribution est « plate » autour de l'espérance. À titre d'exemple, le *kurtosis* de la loi normale est nul, celui de la loi de Laplace est égal à 3 et celui de la loi uniforme est égal à -1.2. Le coefficient de dissymétrie γ_1 est également très intéressant. Comme son nom l'indique, il caractérise la (dis)symétrie de la distribution autour

de son espérance. S'il est positif, la courbe penche à droite et la queue se situe à gauche de l'espérance. À l'inverse, s'il est négatif, la courbe penche vers la gauche et la queue se situe à droite de l'espérance. S'il est nul, la distribution est symétrique (loi normale par exemple).

En général, γ_1 et γ_2 sont estimés à l'aide des formules

$$\gamma_1 \approx \frac{1}{n}\frac{\sum_{i=1}^{n}\left(y_i - \overline{y_n}\right)^3}{(s_n^2)^{3/2}} \quad \text{et} \tag{7.9}$$

$$\gamma_2 \approx n\frac{\sum_{i=1}^{n}\left(y_i - \overline{y_n}\right)^4}{\left(\sum_{i=1}^{n}\left(y_i - \overline{y_n}\right)^2\right)^2} - 3. \tag{7.10}$$

Dans la pratique, dès que n est grand, l'erreur commise est négligeable.

Enfin, nous utilisons également la médiane md et les intervalles interquartile IQR et interdécile IDR. La médiane, tout comme la moyenne, elle permet de « situer » la distribution. Les intervalles interquartile et interdécile expriment eux la notion d'étalement de la distribution autour de cette valeur centrée, tout comme l'écart type. Cependant, la médiane, et les intervalles interquartile et interdécile sont des estimateurs statistiques plus robustes aux valeurs extrêmes.

Le p^{e} centile P_p est défini comme la valeur y_p de Y telle que la probabilité cumulée de y_p est inférieure ou égale à p%, soit :

$$P_p = y_p \quad \text{tel que} \quad \mathcal{P}(Y \leq y_p) \leq p. \tag{7.11}$$

Ainsi, un centile est chacune des 99 valeurs qui divisent les données en cent parts égales, de sorte que chaque partie représente un centième de l'échantillon de population. Pour pouvoir utiliser les centiles, il faut que les données soient triées. Par définition, la médiane est le 50$^{\text{e}}$ centile, c'est-à-dire la valeur qui sépare l'échantillon en deux ensembles de même taille : les valeurs inférieures et les valeurs supérieures. L'intervalle interquartile est la différence entre le 75$^{\text{e}}$ centile et le 25$^{\text{e}}$ centile et représente les 50 % des valeurs de l'échantillon, centrées autour de la médiane. L'intervalle interdécile est la différence entre le 90$^{\text{e}}$ centile et le 10$^{\text{e}}$ centile et représente donc les 80 % des valeurs de l'échantillon, centrées autour de la médiane.

7.4 Développement d'un exemple complet

Dans notre cas, l'espace Ω des possibles est l'ensemble des jeux d'*offsets* différents. En théorie, cet ensemble est infini lorsqu'il n'y a aucune limite imposée sur les valeurs des *offsets* r_i. En ce qui nous concerne, nous ne considérons que les jeux d'*offsets* qui respectent les relations 7.1. Nous avons vu également que parmi les jeux d'*offsets* qui respectent ces relations, il n'y en a que $\prod T_i \,/\, \text{ppcm}\, T_i$ qui sont différents si l'on ne s'intéresse qu'au régime stationnaire du système (car ce faisant, nous pouvons négliger l'influence du *backlog* au démarrage du système).

Ensuite, considérons une variable aléatoire X sur Ω, de loi uniforme dans la mesure où nous supposons que tous les *offsets* sont équiprobables. Si nous diposions de plus d'informations sur les *offsets* (combinaisons plus fréquentes par exemple), nous pourrions choisir une

loi différente pour X qui refléterait ces informations supplémentaires. Considérons de plus la « fonction f » dont la définition est l'algorithme que nous avons présenté au chapitre VI. Pour simplifier, nous ne nous intéressons qu'au minimun et au maximum des délais de chacune des tâches du système et non à leurs lois de distribution complètes. Nous disposons désormais de tous les éléments pour estimer les lois de distribution des variables aléatoires « délai », « délai minimal » et « délai maximal », notées respectivement $Y = f(X)$, $Y_{\min} = f_{\min}(X)$ et $Y_{\max} = f_{\max}(X)$.

En tout, nous présentons les résultats de 18 tirages. Nous avons considéré trois tailles d'échantillons — 10^4, 10^5 et 10^6. Pour chaque, nous avons calculés trois fois les délais avec trois tirages différents d'*offsets* : la première fois, nous avons gardé toute la distribution, la deuxième fois, nous avons gardé le minimun seulement, et la troisième fois, nous avons gardé le maximum. Nous avons doublé ce nombre de configurations en forçant ou non le scénario synchrone parmi les échantillons, pour un total de dix-huit.

Dans les sections suivantes, nous analysons de manière systématique les résultats : loi de distribution des délais, loi de distribution des délais minimaux et loi de distribution des délais maximaux.

7.5 Interprétation des résultats

Dans les trois sections suivantes, nous effectuons plusieurs milliers et même millions de tirages aléatoires parmi les *offsets* possibles pris par les tâches du système temps réel mono-processeur, non-préemptif et composé tel que décrit dans le tableau 5.1 page 54. Nous utilisons ensuite ces *offsets* en entrée de notre algorithme (présenté aux deux chapitres précédents). Dans un premier temps, nous nous intéressons à la loi de distribution des délais des tâches du système au sens le plus général. Puis nous nous intéressons à la loi de distribution des valeurs des délais maximaux. Enfin, nous nous intéressons à la loi de distribution des valeurs des délais minimum.

À chaque étude, nous générons six échantillons nommés S_4, S_5, S_6, N_4, N_5 et N_6. Pour les échantillons S, nous avons forcé le scénario synchrone dans les tirages, pas pour les échantillons N. Le chiffre correspond à la puissance de dix du nombre de tirages dans l'échantillon : l'échantillon S_4 signifie que nous avons forcé le scénario synchrone parmi un échantillon de 10000 tirages.

L'algorithme proposé a été implémenté en Fortran 90 sur une machine virtuelle mono-cœur et 512Mo de RAM tournant sous Linux/Debian et hébergée sur un iMac bi-cœur de 3GHz. En plus du prototype développé, différents scripts écrits en Lua ont été nécessaires pour mettre les résultats sous forme exploitable (voir les détails des outils développés à l'annexe B). Les temps de calcul ont été estimés à l'aide de l'outil `time` ; ils s'échelonnent de l'ordre de 80s pour 10^4 tirages à environ 8000 pour 10^6 tirages, soit environ 2 heures et demi. La progression est presque parfaitement linéaire. Chaque fois, des résultats partiels mettant en avant les grands principes sont présentés sous forme de tableau et de courbes et sont accompagnés de remarques et d'explications. Ils ne sont que partiels afin de ne pas surcharger le manuscrit.

De plus, un tableau récapitulatif est proposé et regroupe différents paramètres statistiques que nous avons calculés chaque fois pour les échantillons S. Dans l'ordre, les paramètres statistiques calculés sur chaque échantillon sont la médiane md_n, les intervalles interquartile et interdécile IQR_n et IDR_n, la moyenne arithmétique $\overline{y_n}$ de l'échantillon et sa variance, la variance $\sigma^2_{s^2_n}$ et sa variance et enfin les coefficients de dissymétrie et d'applatissement γ_1 et γ_2.

Un dernier mot avant de présenter nos résultats. Ces derniers sont tout à fait cohérents entre eux : d'un échantillon à un autre, d'une tâche à une autre et entre les différentes loi de distribution calculées (nous rappelons que les échantillons tirés aléatoirement sont différents à chaque calcul : minimum, maximum et distribution complète des délais). Ceci est très encourageant dans la mesure où la méthode proposée ici est tout à fait nouvelle à notre connaissance et que les outils développés ne sont pas particulièrement optimisés, bien que le choix de Fortran était délibéré à l'époque pour sa manipulation aisée des notations tableaux.

7.5.1 Loi de distribution pondérée des délais

Dans cette section, nous estimons la loi de distribution pondérée des délais pour une instance quelconque de chacune des tâches du système. Pour cela, pour chaque jeu d'*offsets*, chaque tâche et chaque délai subi, nous comptons le nombre total d'instances de la tâche qui subissent ce délai. Nous savons déjà que les délais pour chaque tâche sont bornés par le délai pire cas (pire temps d'exécution moins la durée d'exécution de la tâche), nous savons donc que le calcul est fini (mais long). Les résultats sont pondérés pour tenir compte du poids absolu de chaque délais, quelque soit les *offsets*.

Dans cette section, nous présentons :

- les lois de distribution des délais de la tâche 8 pour tous les échantillons figure 7.1 ;
- les fonctions de répartition des délais de toutes les tâches pour l'échantillon N_6 figure 7.2 ;
- les lois de distribution complètes de toutes les tâches pour les échantillons S_4 et N_6 dans les tableaux 7.1 et 7.2.

Première remarque, la convergence des résultats est très rapide pour les échantillons N. Cela s'observe particulièrement bien sur la figure 7.1. Les trois courbes des échantillons sont pratiquement confondues. Ce résultat conforte notre analyse concernant l'impact mineur du *backlog* à l'instant initial : en régime stationaire, le régime transitoire importe peu.

Deuxième remarque, la convergence des résultats est lente pour les échantillons S. Sur la même figure, nous pouvons observer qu'il existe des pics marqués qui s'estompe avec le nombre grandissant de tirages. Les courbes convergent plus lentement vers les mêmes que pour les échantillons N : la courbe pour l'échantillon S_6 est confondue avec les autres par exemple. L'explication est simple : nous avons pris en compte toutes les combinatoires pour calculer les lois de distribution, or certaines sont grandes, en particulier dans le cas synchrone (5040 pour certaines valeurs des délais, soit l'équivalent de 50 % des tirages de l'échantillon S_4). Dans le cas de l'échantillon S_4, le poids de la valeur maximale pour la tâche 1 est dans un rapport 30 environ à celui de sa valeur minimale 0. Ce rapport passe à plus de quarante mille pour l'échantillon N_4.

délai	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8
0	154225	36543	36415	36398	130442	344513	357200	1630425
1	16151	9235	9419	9462	25971	85079	87794	189595
2	7073	2961	3012	2933	10330	22397	22440	217906
3	7321	3125	3302	3248	10107	25957	26672	265124
4	16444	4872	4806	4771	39294	52749	185634	141484
5	10789	3982	3936	3901	48098	165945	77241	203574
6	4452	2308	2312	2252	17080	68259	15152	364798
7	3276	1757	1751	1681	6267	20731	20574	166099
8	6799	1890	1866	1960	6053	14931	54618	177034
9	10382	2163	2212	2287	46784	55936	44499	178480
10	17534	16274	16129	16210	55973	83479	53310	88939
11	10790	9966	9828	9893	5047	10081	9683	89698
12	1556	931	850	874	3510	8187	6627	133143
13	11402	907	812	916	2723	6734	6146	189448
14	2528	2186	2154	2169	23400	26201	56454	39951
15	2264	2146	2059	2086	16894	39280	7910	30404
16	938	839	841	760	1939	4151	3117	20076
17	344	267	265	261	1117	4047	3882	17594
18	777	637	639	638	4319	5453	32557	62920
19	1502	1466	1464	1484	31577	32500	8667	96974
20	9833	9814	9797	9815	5971	6630	2527	108075
21	12651	12608	12622	12657	1315	1932	1798	46913
22	73	63	90	78	426	819	3839	10411
23	477	466	483	493	17056	17350	23976	11086
24	1615	1614	1619	1619	11127	11291	2036	4742
25	1337	1323	1321	1319	1856	2058	1971	43915
26	447	450	450	454	950	1163	254	41846
27	9	6	12	12	6771	27155	33763	51739
28	433	433	438	437	6770	6843	947	42184
29	1585	1588	1590	1590	910	951	1514	2610
30	5617	5617	5618	5621	2034	2058	613	17268
31	16275	16272	16273	16274	591	653	34207	338
32		1		1	13449	53794	20169	80883
33	720	720	721	720	7	15	9	82196
34	2448	2448	2448	2448	580	576	1155	41542
35	625	624	624	624	1536	1536	1921	40972
36	240	240	240	240	960	960	20160	32
37					1	20161		10
38	240	240	240	240	1	1	960	972
39	1200	1200	1200	1200	960	960	2880	42250
40	1440	1440	1440	1440	3024	3024	144	16280
41					240	240	240	
43	720	720	720	720				2880
44	5040	5040	5040	5040	240	240	480	480
45					960	960	1440	240
46					720	720		40320
48							720	720
49					720	720	5040	1440
50					5040	5040		
53								5040

Tableau 7.1 Distribution des délais des tâches pour l'échantillon S_4.

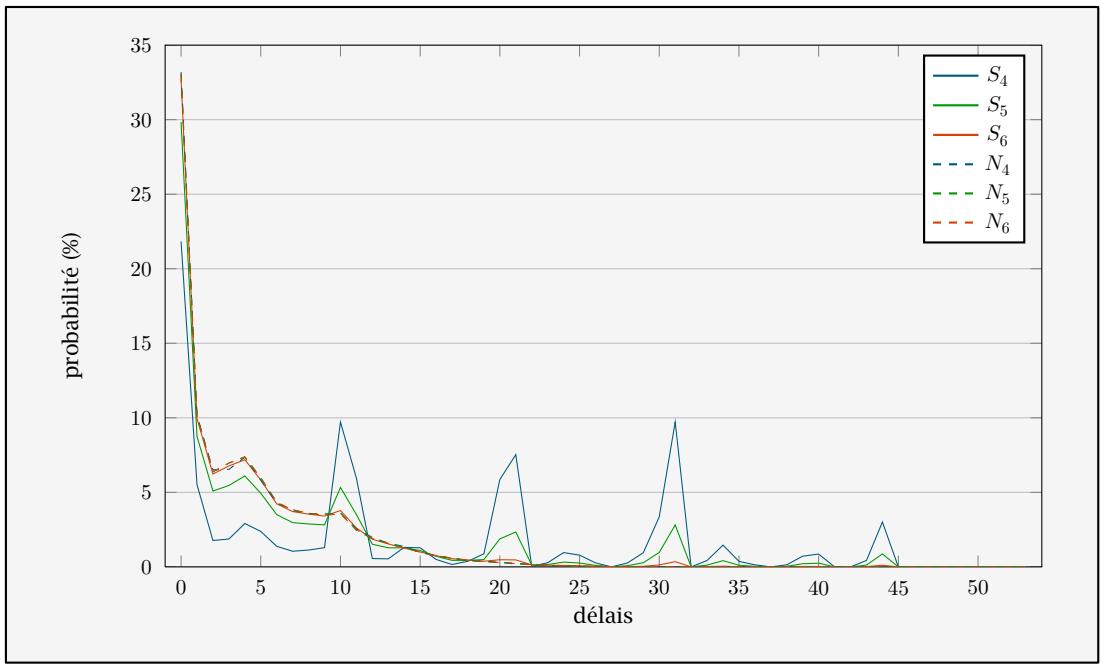

Figure 7.1 Distribution des délais de τ_8 pour tous les échantillons.

Ce résultat est très intéressant : l'influence du scénario pire cas n'est pas à négliger dans l'ensemble et en particulier si le système réel n'évolue pas longtemps. Dans le cas de simulations, il faut soit effectuer des simulations longues (particulièrement si l'on souhaite modéliser le phénomène de dérive d'horloge), soit, paradoxalement, empêcher l'apparition du scénario synchrone pour ne pas influencer trop fortement les résultats. Le pire cas est rare, mais à une grande influence localement.

Troisième remarque, les résultats obtenus pour les tâches 2, 3 et 4 sont très similaires, voire égaux si l'on prend en compte qu'il s'agit d'estimations statistiques. C'est une très bonne nouvelle ! En effet, les trois tâches ont les mêmes caractéristiques temporelles : durées d'exécution et périodes égales, il est donc normal qu'elles aient la même loi de distribution ce qui se vérifie très bien sur la figure 7.2 et dans les tableaux 7.1, 7.2 et 7.3.

Quatrième remarque, les fonctions de répartition obtenues sont assez lisses pour les grands échantillons et très semblables (les courbes se tiennent en 10 points de pourcentage). Elles croissent rapidement au début et ralentissent assez vite vers la fin. D'une manière générale, la queue des distributions ne peut pas être négligées.

Les raisons d'un délai faible pour une tâche sont d'une part un *backlog* faible à l'activation. Comme le système n'est pas extrêmement chargé, les petites valeurs de *backlog* sont plus nombreuses. Si pour le scénario synchrone, le maximum de *backlog* est de 46, il oscille généralement entre 15 et 20 pour les échantillons N et ce de manière très ponctuelle. D'autre part, plusieurs tâches ont des durées d'exécution faibles, cela augmente les fortes chances pour une tâche d'être activée en même temps que l'une de ces tâches « rapides ». Cela explique la croissance rapide des fonctions de répartition aux faibles valeurs.

À l'inverse, les plus grandes valeurs des délais sont plus difficiles à obtenir. Le système n'étant pas très chargé, le *backlog* n'est jamais très grand. D'autre part, pour obtenir des valeurs plus élevées de délais, il faut plus de tâches pour y contribuer. Or, il est aussi plus rare

délai	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8
0	4167545	1519972	1520077	1521890	5863328	17057652	18168234	68828935
1	1124130	466324	465500	467955	1510153	3356223	3563280	10862482
2	723070	294072	293834	294704	1026175	2248912	2267877	13693036
3	751599	320329	320809	320527	1067478	2566095	2730845	14484626
4	809668	338391	337693	337965	1198583	2461112	2829874	11966870
5	616757	273259	273690	274273	1054084	2592902	1592102	11206632
6	414834	200700	199142	200945	708403	1653823	1508748	8173090
7	342515	175129	174971	174140	647618	2018522	2049514	8438871
8	310381	166658	166553	165913	633601	1469263	1369224	8555711
9	298485	158689	158960	159250	636079	1491268	1451124	7637347
10	279229	163768	161756	163831	594537	1353149	1307001	6970142
11	191974	114800	113965	114831	409256	889408	855004	4891421
12	147373	88480	87575	87716	363621	786966	634610	5316687
13	117539	72784	72610	72087	286951	668802	611559	3754279
14	92111	58029	57863	58134	233783	494584	412282	2760297
15	69609	45666	45437	45411	188083	389765	320319	2868736
16	53759	34988	34589	34823	146866	330984	300304	1978546
17	37281	26531	26233	25908	118269	399034	381725	1723638
18	28181	20682	20469	20711	98501	208430	173245	1511346
19	22532	16458	16341	16537	75419	160652	137332	1204163
20	18063	13548	12997	13570	55738	125338	106600	940635
21	12882	9939	9290	9794	44314	99303	86183	643893
22	9015	6826	6868	6605	38291	82610	52422	681860
23	6143	4824	4776	4863	24539	54497	44812	366749
24	4411	3487	3379	3417	17630	36098	28287	275696
25	3023	2440	2239	2489	13446	27080	22044	268804
26	1956	1526	1512	1581	10200	23294	19785	166010
27	1115	969	966	984	7538	26366	23668	130410
28	764	687	662	653	5660	11267	8139	102166
29	541	437	476	480	3621	7774	5894	82358
30	390	312	302	382	2477	5954	4243	50289
31	294	321	187	310	1879	4241	2950	32160
32	127	111	119	123	1363	3042	1233	33067
33	89	76	76	90	680	1183	798	13451
34	48	26	54	46	455	756	465	9913
35	31	25	34	48	295	541	374	7946
36	6	11	14	8	230	426	284	4152
37	6	5		3	219	309	112	2673
38	6	10	1	8	85	125	95	1773
39					38	85	44	1538
40					31	44	27	603
41					19	28	9	351
42					10	10	3	305
43					1	3	6	78
44					3			103
45								50
46								6
47								6

Tableau 7.2 Distribution des délais des tâches pour l'échantillon N_6.

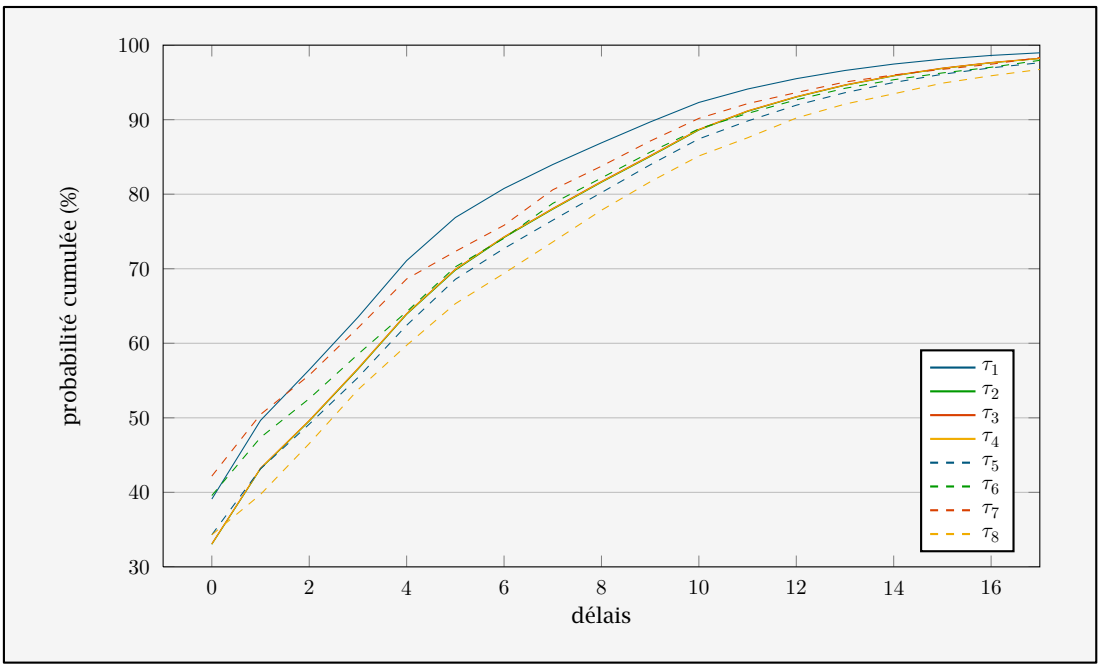

Figure 7.2 Fonction de répartition des délais de τ_1 à τ_8, pour l'échantillon N_6.

	md_n	IQR_n	IDR_n	$\overline{y_n}$	$\sigma^2_{\overline{y_n}}$	s^2_n	$\sigma^2_{s^2_n}$	γ_1	γ_2
τ_1	2	11	29	7.8866	0.0003	121.1285	0.1413	1.4990	1.3673
τ_2	10	20	31	13.9308	0.0010	165.6024	0.2041	0.6139	-0.7540
τ_3	10	20	31	13.9282	0.0010	166.0856	0.2045	0.6125	-0.7614
τ_4	10	20	31	13.9379	0.0010	165.8212	0.2041	0.6118	-0.7588
τ_5	7	13	24	9.6964	0.0002	103.1725	0.0768	1.4213	2.1225
τ_6	5	10	27	8.4051	0.0001	104.1310	0.0340	1.5659	1.8972
τ_7	4	10	27	7.9553	0.0001	101.1368	0.0327	1.6102	1.9692
τ_8	5	12	26	8.3130	0.0000	110.3958	0.0095	1.5772	1.9187
τ_1	2	6	12	4.5419	0.0000	49.7734	0.0192	2.5931	8.1156
τ_2	4	10	20	6.9579	0.0001	83.8282	0.0642	1.8569	3.3009
τ_3	4	10	20	6.9580	0.0001	83.7812	0.0643	1.8589	3.3104
τ_4	4	10	20	6.9605	0.0001	83.9860	0.0645	1.8576	3.2948
τ_5	4	9	15	5.7807	0.0000	52.4898	0.0109	2.0979	6.2983
τ_6	3	8	13	5.0628	0.0000	47.8146	0.0039	2.2571	6.6530
τ_7	2	7	13	4.7007	0.0000	45.4189	0.0037	2.3598	7.1383
τ_8	3	8	14	5.5049	0.0000	49.3863	0.0008	2.0767	5.8159
τ_1	2	5	10	3.4725	0.0000	22.0614	0.0004	2.2206	8.0740
τ_2	3	7	12	4.4763	0.0000	31.3100	0.0018	2.0741	6.7507
τ_3	3	7	12	4.4757	0.0000	31.2821	0.0018	2.0780	6.7771
τ_4	3	7	12	4.4692	0.0000	31.3445	0.0018	2.0806	6.7759
τ_5	3	7	12	4.4933	0.0000	29.1530	0.0003	1.6978	4.4624
τ_6	2	7	11	4.1212	0.0000	27.5995	0.0001	1.7502	4.3550
τ_7	1	6	11	3.7833	0.0000	25.7007	0.0001	1.8589	4.8729
τ_8	3	8	13	4.8073	0.0000	31.7171	0.0000	1.5314	3.1042

Tableau 7.3 Paramètres statistiques des lois de distribution des délais de toutes les tâches pour les échantillons S_4, S_5 et S_6.

que beaucoup de tâches soient activées en même temps. Tout ceci explique un ralentissement de la croissance de la courbe aux grandes valeurs. Dans l'ensemble, le *backlog* a un véritable effet moyennant et lissant sur les lois de distribution.

Cinquième remarque, il semble exister des valeurs privilégiées en fonction des tâches. Cela s'observe mieux sur la figure 7.1 pour les échantillons S. Pour la tâche 1 par exemple dans le tableau 7.2, les délais 1, 4 et 5 ont des occurrences de plus de 500000. Il y a plusieurs causes : la combinatoire élevée, le *backlog*, l'activation fréquente en même temps que d'autres tâches et la fréquence plus élevée des tâches de faible période. Nous avons déjà mentionné l'effet des deux premières. Les deux autres s'expliquent tout aussi bien.

En fonction de la période des tâches, certaines vont se retrouver activées ensembles plus ou moins souvent. Deux tâches dont les périodes sont identiques sont soit toujours activées ensemble, soit jamais. C'est le cas pour les tâches 2, 3 et 4. La probabilité d'un délai de 10 est faible, de 20 encore plus faible (trois tâches activée parmi les quatre de durée d'exécution 10) et un délai de 30 est marginal (les quatres tâches activées). En revanche, si les périodes sont différentes, l'influence d'une tâche sur une autre dépend du pgcd entre les tâches. Plus celui-ci est grand, moins il y a de chance pour qu'un *offset* les fasse se rencontrer et vice-versa. Le cas extrême est celui des périodes premières entre elles. Dans ce cas, quelque soit l'*offset*, les deux tâches seront activées au moins une fois en même temps au cours de l'évolution du système.

Dans l'exemple proposé, il n'y a pas de tâches avec des périodes premières entre elles. Le plus petit pgcd est entre les tâches 6/7 et 8 et vaut 2. Cela explique en partie les grandes valeur observées pour les tâches 6 et 7 au délai 1 et celles pour la tâche 8 au délai 4.

Sixième remarque, il y a un gros pic en 0, entre 30 % et 40 % de la distribution. Ceci est tout à fait compréhensible : quelque soit le tirage et quelque soit l'instant d'activation, toute instance d'une tâche peut se retrouver la première dans la file. Comme évoqué, l'influence du *backlog* est difficile à évaluer, mais de toute évidence, le système est souvent au repos puisque nombreuses sont les possibilités pour les tâches de ne pas subir de délai.

Septième remarque, les lois ne sont pas symétriques, mais penchées à gauche avec une queue à droite ($\gamma_2 > 0$), pas très piquée vers la valeur médiane/moyenne ($\gamma_1 > 0$) et le pire cas n'apparaît pas souvent : pas une fois en plusieurs millions de tirages en fait. En revanche, comme le suggère les courbes de la figure 7.2 et le tableau 7.3, le poids de la queue de la distribution n'est pas négligeable. En effet, si 50% et 90% des échantillons se retrouvent dans des intervalles de petite amplitude autour de la médiane (valeurs IQR_n et IDR_n relativement faibles), les résultats suggèrent que pour aller chercher les 10% restant, il faut multiplier l'intervalle interdécile par deux ou trois. Par exemple, le 99$^{\mathrm{e}}$ décile de la tâche 1 est de 20 contre 10 pour le 90$^{\mathrm{e}}$.

7.5.2 Loi de distribution des délais maximaux

Dans cette section, nous estimons la loi de distribution des valeurs des délais maximaux pour chacune des tâches du système. Pour cela, pour chaque jeu d'*offsets*, chaque tâche et chaque délai maximal subi, nous ajoutons 1. Ici, il ne s'agit pas d'évaluer le poids de chaque

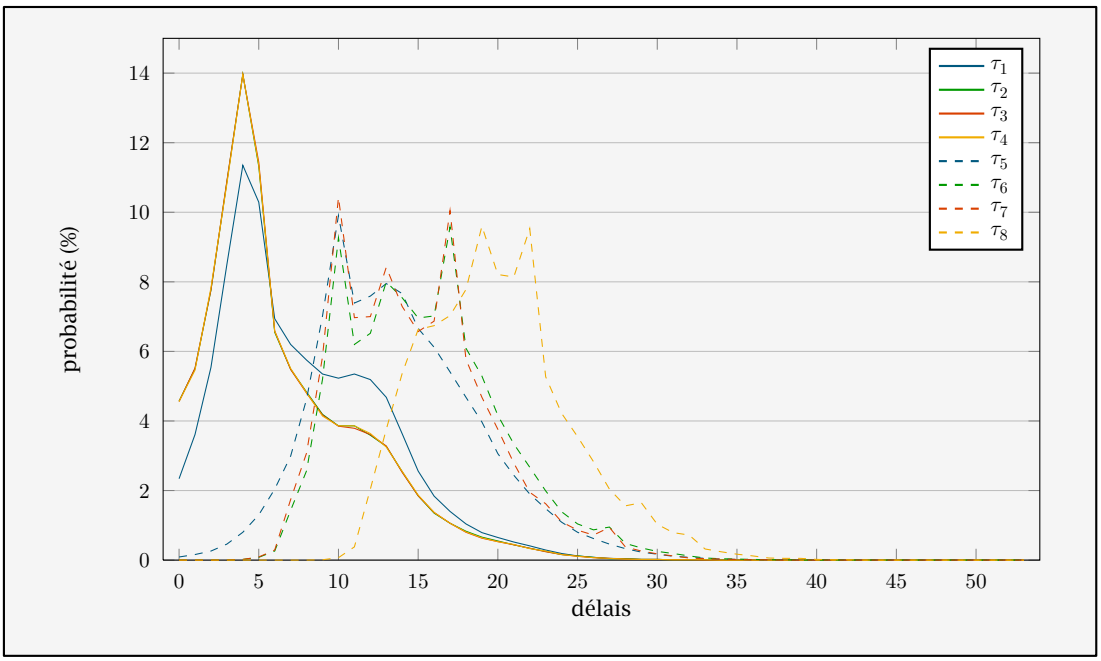

Figure 7.3 Loi de distribution des valeurs des délais maximaux de toutes les tâches pour l'échantillon S_6.

valeurs dans le cas général comme à la section précédente, mais plutôt de cartographier les délais maximaux des tâches, le but étant de voir si les délais maximaux sont globalement proches des cas pires. Nous nous concentrons sur les échantillons S dans la mesure où nous voulons évaluer les lois de distributions par rapport aux cas pires, d'autre part, ce scénario ne compte ici que pour 1 et a moins d'influence que dans la section précédente.

Dans cette section, nous présentons :

- les fonctions de répartition des délais maximaux de toutes les tâches pour les échantillons S_4 et S_6 figure 7.4 ;
- les lois de distribution complètes des délais maximaux de toutes les tâches pour les échantillons S_4 et S_6 dans le tableau 7.4 et les figures 7.3 et 7.4.

Première remarque, de nouveau, les résultats pour les tâches 2, 3 et 4 sont (statistiquement) identiques, ce qui est toujours aussi rassurant. Ce sont les courbes les plus à gauche, décalées vers les faibles valeurs des délais maximaux. Cela signifie que ces tâches ne sont pas celles qui subiront le plus de délais en moyenne.

Deuxième remarque, les courbes sont toutes semblables, mais plus ou moins décalées vers la droite : peu de forts délais par rapport au cas pire. Dans l'ordre de la gauche vers la droite, les courbes sont celles de la tâche 2, 3 et 4 (les unes sur les autres), puis la tâche 1, la tâche 5, la tâche 7, la tâche 6 et enfin la tâche 8. Cet ordre est à rapprocher de la période des tâches (facteur dominant) et de la durée d'exécution. L'influence de cette dernière est moindre, mais il peut s'observer entre les courbes de τ_6 et τ_7 : les deux tâches sont très semblables et ne diffèrent que d'une unité sur leurs durées d'exécution à la faveur de τ_7. Il est normal que la courbe de τ_6 soit très légèrement décalé de celle de τ_7 vers les valeurs plus grandes puisque lorsque τ_7 est devant elle, τ_6 est gêné plus longtemps que τ_7 lorsque τ_6 est devant elle. Plus la période d'une tâche est faible, plus elle est exécutée souvent et plus elle

max.	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8
0	224	447	470	471	4				23417	45664	45584	45593	898			
1	358	525	554	547	19				36111	55045	55211	54629	1622			
2	519	807	733	805	27				55375	77565	77673	78264	2623			
3	863	1056	1094	1080	45		1		84969	108746	109010	108201	4563	34	39	
4	1110	1425	1377	1386	84	6	2		113518	139680	139624	139575	8049	241	247	
5	1090	1117	1151	1161	133	7	12		102913	113238	114372	113630	13014	801	876	
6	715	670	676	595	183	24	32		69447	65521	65940	65843	20355	2552	2892	
7	626	539	560	586	294	151	188		62034	54876	54795	55137	29895	14252	17385	
8	613	514	496	471	472	251	309		57473	48338	48021	48141	46329	25736	30765	
9	482	410	412	422	682	501	581	1	53466	41866	41727	41377	70211	51959	58507	51
10	519	397	397	388	995	911	1057	12	52317	38636	38529	38582	99145	92665	104344	664
11	565	374	373	366	798	611	750	26	53497	38600	37913	38554	73934	61954	69734	3758
12	505	357	376	342	759	713	671	208	51939	35895	36098	36416	75892	65238	69977	20383
13	487	321	311	333	774	813	823	361	46778	32747	32842	32509	79462	80001	84230	37579
14	347	265	262	272	738	721	723	548	36340	25410	25158	25519	76584	75403	72879	53692
15	233	183	195	187	685	684	585	666	25554	18536	18450	18656	66647	69613	65660	66335
16	194	135	119	137	612	705	740	682	18408	13501	13676	13654	61164	70244	68696	67394
17	111	87	98	89	533	1006	998	689	13971	10571	10627	10577	54062	95820	100487	70270
18	109	73	71	86	492	607	571	801	10392	8306	8006	8077	46753	60928	57747	77808
19	75	65	73	60	385	495	481	952	7852	6562	6296	6440	39731	52893	46808	96023
20	64	72	61	54	325	421	359	800	6546	5474	5345	5344	30607	41624	37643	82086
21	50	48	35	41	237	341	283	853	5181	4379	4376	4458	24405	33390	27913	81435
22	47	39	30	38	160	257	191	943	4073	3385	3400	3334	19033	26733	19532	95258
23	28	18	26	28	149	165	162	546	2941	2454	2435	2552	14798	19940	16229	52746
24	18	25	21	24	99	144	104	404	1934	1626	1616	1639	10908	13867	11016	42249
25	14	9	10	6	84	119	76	367	1202	1156	1098	1065	7981	10366	8572	35530
26	12	4	4	6	62	95	71	255	802	769	723	734	6212	8735	7180	28281
27	9	7	5	5	53	97	109	200	509	437	504	535	4578	9539	9377	20351
28	3	2	3	4	36	51	33	158	369	355	314	317	3334	4775	3830	15590
29	3	2	3	3	25	32	36	139	200	217	246	202	2285	3521	2592	16475
30	3	2	2	2	22	30	16	111	167	153	151	168	1704	2452	1844	10290
31	1	1		1	12	15	16	86	112	105	94	105	1114	1870	1198	7894
32	1			1	10	12	10	90	83	85	74	67	758	1185	622	7204
33				1	6	4	3	29	48	51	35	54	518	624	424	3207
34	1	1			3	2	4	21	36	27	21	23	313	371	253	2423
35		1	1			3		18	10	13	9	14	191	258	206	1686
36		1		1	1	2		10	14	8	4	7	115	165	131	1219
37						1	1	10		1	1	3	79	97	72	647
38						1	1	6	1		1	3	55	61	45	457
39					1			5		1		1	38	40	25	472
40						1							20	24	14	214
41								1					10	17	4	141
42								1					9	10	3	107
43															1	32
44	1	1	1	1					1	1	1	1	1	1		28
45																13
46																5
47																1
48																1
49							1								1	
50					1	1							1	1		
53								1								1

Tableau 7.4 Distribution des délais maximaux de toutes les tâches pour les échantillons S_4 et S_6.

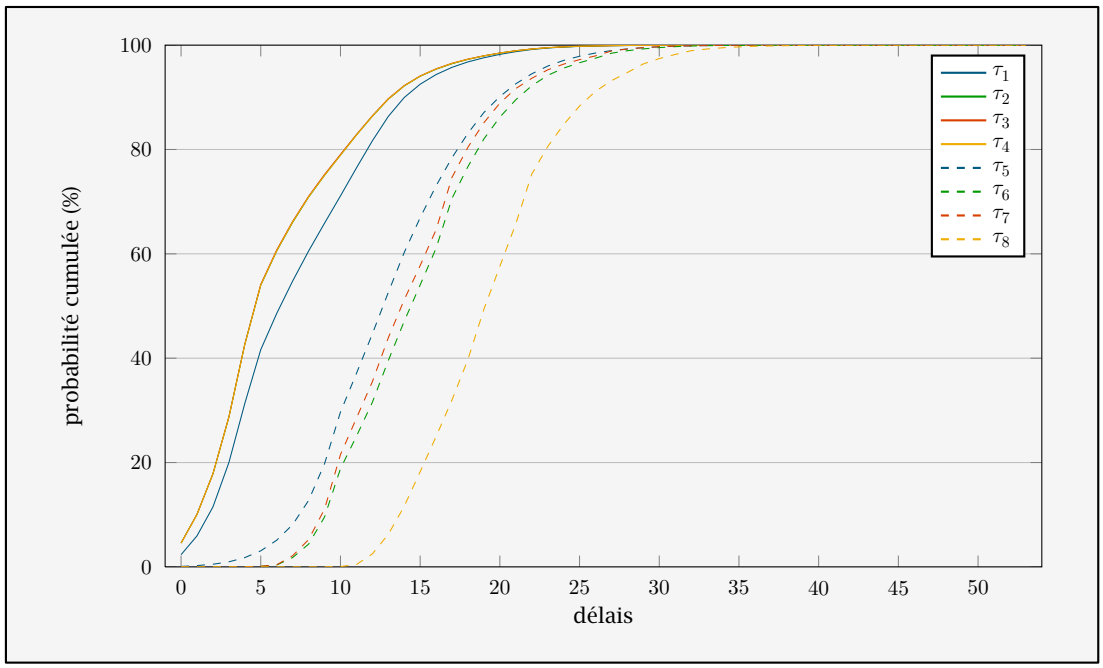

Figure 7.4 Fonction de répartition des valeurs des délais maximaux de toutes les tâches pour l'échantillon S_6.

rencontre de trafic. Statistiquement, les tâches fréquentes ont plus de chance de subir des délais maximaux plus importants que les autres.

Troisième remarque, les valeurs du cas pire ne sont pas si eloignées des observations des valeurs des délais maximaux. De plus, les courbes ont des queues larges : l'intervalle interdécile est de 12 pour la tâche 8 et sont 99$^{\text{e}}$ décile est de 33. Nous remarquons une ou deux valeurs plus ou moins privilégiées (même trois pour les tâches 6 et 7), liées à la combinatoire des tâches.

Quatrième remarque, les résultats obtenus pour les délais maximaux sont cohérents avec les résultats précédents et expliquent pourquoi la ditribution générale est si déplacée vers les valeurs faibles. En effet, un délai maximal faible implique que tous les délais pour ces *offsets*–là sont faibles.

Cinquième remarque, sans forcer le cas synchrone, le maximum observé des délais est plus faible, mais se rapproche du cas pire. En revanche, nous ne l'avons jamais observé. Cela semble confirmer que le cas pire ne survient que lors du cas synchrone. Cela dit, nous n'avons tiré que quelques millions de jeux d'*offsets* parmi les $5.76 \cdot 10^{11}$, soit moins de 0.001% des *offsets*. Toute conclusion sur ces seules bases serait hâtive.

Une conclusion que nous pouvons faire néanmoins : en pratique, les systèmes temps réel peuvent s'approcher du cas pire et parfois relativement près. Les approches sûres restent nécessaires pour les systèmes critiques, comme c'est le cas des réseaux avioniques ou du réseau de commande d'une centrale nucléaire (par exemple). Pour les autres, une borne assortie d'une probabilité de la dépasser pourrait être suffisant (voir [52] par exemple).

	md_n	IQR_n	IDR_n	$\overline{y_n}$	$\sigma^2_{\overline{y_n}}$	s^2_n	$\sigma^2_{s^2_n}$	γ_1	γ_2
τ_1	7	7	12	7.8063	0.0025	25.2028	0.1833	0.8943	0.8859
τ_2	5	6	12	6.6563	0.0025	24.5924	0.2197	1.1902	1.6345
τ_3	5	6	13	6.6189	0.0024	24.0479	0.2008	1.1509	1.4727
τ_4	5	6	13	6.6202	0.0025	24.7118	0.2203	1.1865	1.6083
τ_5	13	7	12	13.7261	0.0026	25.8613	0.1830	0.6109	0.7372
τ_6	15	6	12	15.2997	0.0023	23.4417	0.1432	0.6711	0.6070
τ_7	14	7	12	14.7830	0.0022	22.4847	0.1346	0.7172	0.6628
τ_8	20	6	12	19.9436	0.0022	21.6318	0.1280	0.6778	0.7355
τ_1	7	7	12	7.7960	0.0002	24.9241	0.0154	0.8202	0.4844
τ_2	5	6	12	6.6407	0.0002	24.2752	0.0190	1.1316	1.2254
τ_3	5	6	12	6.6237	0.0002	24.2426	0.0196	1.1536	1.3270
τ_4	5	6	13	6.6265	0.0002	24.2925	0.0194	1.1393	1.2872
τ_5	13	7	12	13.6951	0.0003	25.7016	0.0165	0.5574	0.5018
τ_6	15	7	12	15.3021	0.0002	23.3721	0.0131	0.6415	0.3950
τ_7	14	7	12	14.7896	0.0002	21.9577	0.0117	0.6791	0.4310
τ_8	20	6	12	19.9392	0.0002	21.6116	0.0120	0.6576	0.5691
τ_1	7	7	13	7.8047	0.0000	25.0970	0.0015	0.8120	0.4555
τ_2	5	6	13	6.6355	0.0000	24.3310	0.0019	1.1286	1.1992
τ_3	5	6	13	6.6179	0.0000	24.1788	0.0019	1.1289	1.1921
τ_4	5	6	13	6.6342	0.0000	24.2875	0.0019	1.1279	1.1991
τ_5	13	7	12	13.7126	0.0000	25.7109	0.0017	0.5620	0.5089
τ_6	15	7	12	15.2991	0.0000	23.3529	0.0013	0.6362	0.3671
τ_7	14	7	12	14.8000	0.0000	21.9714	0.0012	0.6793	0.4391
τ_8	20	5	12	19.9453	0.0000	21.5601	0.0012	0.6435	0.5434

Tableau 7.5 Paramètres statistiques des lois de distribution des délais maximaux de toutes les tâches pour les échantillons S_4, S_5 et S_6.

7.5.3 Loi de distribution des délais minimaux

Dans cette section, nous estimons la loi de distribution des valeurs des délais minimaux pour chacune des tâches du système. Pour cela, pour chaque jeu d'*offsets*, chaque tâche et chaque délai minimal subi, nous ajoutons 1. Ici, il ne s'agit pas d'évaluer le poids de chaque valeurs dans le cas général comme à la première section, mais plutôt de cartographier les délais minimaux des tâches.

Avec ces résultats, nous voulons mettre en avant que l'ensemble des jeux d'*offsets* où les tâches subiront systématiquement des délais est non vide. Cela est d'autant plus vrai que le système sera plus chargé. Sans être extrême, la charge de notre système est supérieur à 70 %, bien au-delà des charges rencontrées usuellement dans l'industrie, plus proche de 30 à 50 %.

Les résultats sont consultables dans les tableaux 7.6 et 7.7 pour les échantillons S_4 et S_6. Le scénario synchrone n'a aucune incidence dans ce cas.

7.6 Conclusions

Dans ce chapitre, nous avons retrouvé un résultat démontré indépendamment par Goo-

min.	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6	τ_7	τ_8
0	9554	6754	6747	6690	7379	8698	8996	10^4	957726	674993	675288	674036	740495	864450	899791	10^6
1	133	380	389	390	628	353	321		12629	38226	38210	38123	65912	35693	33529	
2	89	360	333	313	636	318	326		8764	33879	33828	33717	65341	33091	33436	
3	65	352	345	340	711	329	357		6739	35011	34892	34837	66965	33576	33244	
4	44	345	337	341	446	302			4706	32179	32062	32409	41085	33190		
5	35	322	297	316	98				3443	30599	30437	30513	10234			
6	33	263	307	319	62				2353	28251	28475	28898	5492			
7	27	266	289	296	29				1771	28288	28530	28790	3369			
8	16	262	249	292	11				1174	28271	28093	28054	1107			
9	4	284	275	266					619	27239	27462	27383				
10		97	104	113					64	11026	11136	11092				
11		66	89	82					11	7185	7273	7366				
12		52	53	48					1	5411	5244	5428				
13		55	52	42						4777	4717	4685				
14		34	39	38						3916	3880	3861				
15		28	32	35						3184	3136	3158				
16		28	16	29						2382	2295	2460				
17		17	18	16						1836	1727	1835				
18		16	12	13						1291	1262	1281				
19		3	7	10						830	777	821				
20		6	3	4						448	472	475				
21		2	2	3						292	316	307				
22		2		2						183	176	155				
23		1	4							110	113	135				
24		2		1						75	88	83				
25		2								59	55	49				
26		1	1							39	33	23				
27				1						13	16	14				
28										6	5	7				
29										1	2	4				
30												1				

Tableau 7.6 Distribution des délais minimaux de toutes les tâches pour les échantillons S_4 et S_6.

sens et Cucu-Grosjean dans [28] qui borne le nombre de jeux d'*offsets* différents pour un ensemble de tâches temps réelles périodiques de périodes T_i : $\prod_i T_i / \operatorname{ppcm} T_i$. Ce rapport peut prendre de très grandes valeurs, en particulier si les périodes sont harmoniques comme c'est le cas des VL dans un réseau AFDX.

Nous avons proposé d'utiliser une méthode de type Monte Carlo pour approcher différentes lois de distribution de tâches temps réel. Pour cela nous avons rappelé les résultats importants de ce genre de méthodes : savoir que la loi de distribution sur un échantillon *représentatif* tend en loi vers celle de la population avec la taille de l'échantillon. En particulier, nous avons caractérisé sur un exemple la moyenne et l'écart type du délais maximal d'une tâche, tous jeux d'*offsets* confondus, et l'avons comparé à son délais pire cas calculé par notre algorithme du chapitre précédent (grâce au jeu *offsets* synchrones). Dans la limite de nos hypothèses, cela nous a permit de conclure qu'en règle générale, le délais pire cas est rare et peu représentatif du comportement moyen des systèmes temps reél, mais qu'il n'est pas négligeable dans le cas de systèmes critiques, comme en aéronautique par exemple.

	md_n	IQR_n	IDR_n	$\overline{y_n}$	$\sigma^2_{\overline{y_n}}$	s^2_n	$\sigma^2_{s^2_n}$	γ_1	γ_2
τ_1	0	0	0	0.1408	0.0001	0.6314	0.0022	6.8964	51.9887
τ_2	0	3	7	1.8604	0.0012	11.9859	0.1011	2.1761	5.0402
τ_3	0	3	7	1.8743	0.0012	11.8957	0.0899	2.0812	4.3554
τ_4	0	3	7	1.9305	0.0012	12.2633	0.0941	2.0523	4.2602
τ_5	0	1	3	0.6970	0.0002	1.8656	0.0020	2.0521	3.7935
τ_6	0	0	1	0.3184	0.0001	0.8404	0.0007	2.9542	7.6518
τ_7	0	0	1	0.2044	0.0000	0.4420	0.0002	3.3258	9.9744
τ_8	0	0	0	0.0000	0.0000	0.0000	nan	nan	nan
τ_1	0	0	0	0.1275	0.0000	0.5508	0.0002	7.2920	60.0508
τ_2	0	3	7	1.8768	0.0001	12.0048	0.0094	2.1074	4.5472
τ_3	0	3	7	1.8945	0.0001	12.1740	0.0096	2.0945	4.4465
τ_4	0	3	7	1.8870	0.0001	12.0451	0.0093	2.0871	4.3812
τ_5	0	1	3	0.6785	0.0000	1.8273	0.0002	2.1496	4.3962
τ_6	0	0	1	0.3371	0.0000	0.8978	0.0001	2.8629	7.0510
τ_7	0	0	0	0.1973	0.0000	0.4222	0.0000	3.3863	10.4492
τ_8	0	0	0	0.0000	0.0000	0.0000	nan	nan	nan
τ_1	0	0	0	0.1287	0.0000	0.5578	0.0000	7.2668	59.4665
τ_2	0	3	7	1.8853	0.0000	12.1101	0.0010	2.1033	4.4906
τ_3	0	3	7	1.8815	0.0000	12.0527	0.0009	2.1003	4.4858
τ_4	0	3	7	1.8937	0.0000	12.1449	0.0010	2.0938	4.4458
τ_5	0	1	3	0.6784	0.0000	1.8166	0.0000	2.1257	4.2239
τ_6	0	0	1	0.3354	0.0000	0.8888	0.0000	2.8632	7.0643
τ_7	0	0	1	0.2001	0.0000	0.4264	0.0000	3.3563	10.2489
τ_8	0	0	0	0.0000	0.0000	0.0000	nan	nan	nan

Tableau 7.7 Paramètres statistiques des lois de distribution des délais minimaux de toutes les tâches pour les échantillons S_4, S_5 et S_6.

Remarque Ces résultats ont été présentés lors de la conférence EUCASS'2011 et ont été sélectionnés parmi l'ensemble des articles de la conférence pour être publiés dans [49].

Conclusions et perspectives VIII

8.1 Conclusions des travaux présentés

Dans le contexte de réseaux AFDX critiques, nous nous sommes intéressés aux lois de distribution des flux de trames émises par les flux de données appelés VL. Pour cela, nous nous sommes ramenés à un problème d'analyse de systèmes temps réel dont les tâches et les instances représentent les VL et les trames. Lors de ces travaux, nous nous sommes intéressés aux lois de distribution des temps de réponese (ou des délais) de tâches d'un système temps réel, monoprocesseur, non–préemptif, à priorité unique, gérée en FIFO.

Dans un premier temps, nous avons proposé un algorithme pour le calcul de la loi de distribution exacte des temps de réponse d'un ensemble d'instances réveillées simultanément. Pour cela, nous avons introduit deux notions importantes : les classes d'équivalence et les motifs. Cela nous a permi de réduire la complexité du problème en $\mathcal{O}(n!)$ à une complexité moindre, mais qui dépend des valeurs des données et non plus seulement de leur taille. Si toutes les tâches ont la même durée d'exécution C_i, alors la complexité est en $\mathcal{O}\left(n^2\right)$; si toutes les tâches ont des durées d'exécution différentes, alors la complexité est en $\mathcal{O}(n \cdot 2^n)$. D'autre part, et sous les mêmes conditions, la taille de la table des motifs varie entre $\mathcal{O}(n)$ et $\mathcal{O}(2^n)$. Entre les deux, cela dépend du nombre de tâche par classe d'équivalence ; si elles sont équilibrées, alors la taille de la table est en $\mathcal{O}\left((n \,/\, k+1)^k\right)$ où k est le nombre de classes.

Ensuite et pour résoudre le problème d'explosion combinatoire lié aux multiples combinaisons des instances dans la file d'attente, nous avons introduit une notation symbolique et efficace des traces des activations des tâches au cours du temps. Nous l'avons utilisée pour le calcul de différentes propriétés entourant les temps de réponse des tâches, en particulier les temps de réponse maximaux. De plus, si l'on sait que le jeu d'*offsets* conduit au pire cas (théorème de l'instant critique par exemple), alors les maxima calculés sont les temps de réponse pire cas exacts des tâches. Nous avons également présenté notre algorithme original et efficace du calcul de la loi de distribution exacte des temps de réponse (ou des délais) des tâches et l'avons illustré sur un exemple. Cet algorithme a été implémenté en Fortran 90 (voir annexe B).

Enfin, nous avons mis ces résultats en application en mettant en œuvre une méthode de type Monte Carlo pour approcher différentes loi de distribution des délais des tâches. Après avoir rappelé les résultats importants de ce type de méthodes statistiques, nous avons étudié sur l'exemple support la loi de distribution pondérée des temps de réponse des tâches du système, ainsi que celles des temps de réponse minimaux et maximaux. Dans le cadre de nos hypothèses, cela nous a permis de conclure qu'en règle générale, le délai pire cas est rare et peu représentatif du comportement moyen des systèmes temps reél où les délais sont significativement plus faibles en majorité. Cependant, il n'est pas négligeable dans le contexte de systèmes critiques où chaque anomalie doit être signalée (au minimum), comme en aéronautique par exemple.

Les méthodes disponibles à l'heure actuelle s'utilisent à différentes étapes de la conception et de l'implémentation des systèmes. En amont, comme aide à la conception, ou *a posteriori* comme outil de validation, vérification, preuve et/ou certification. La méthode décrite dans cette thèse appartient plutôt à la première catégorie, mais pourrait trouver un intérêt dans la seconde.

Aujourd'hui cependant, et en l'état, il n'est pas envisageable d'utiliser ces travaux sur des configurations de tailles industrielles réalistes avec de multiples commutateurs, reliés les uns aux autres. Il y a deux problèmes.

Premier problème, la précision de la distribution des délais en sortie de notre algorithme dépend du modèle de celle de l'entrée. Dans une configuration à un seul commutateur, le modèle d'entrée est exact (sous les hypothèses faites). Dans une configuration avec plusieurs commutateurs, la distribution des délais en sortie d'un commutateur sert d'entrée au suivant. De fait, la précision se perd au fur et à mesure des étages traversés.

Deuxième problème, le temps de calcul. Le nombre de combinaisons des *offsets* à prendre en compte est extrêment grand et la convergence, plutôt lente, en $\mathcal{O}\left(\sqrt{n}\right)$. Gagner un ordre de grandeur en précision demande de multiplier l'échantillon par 100, et donc, d'autant le temps de calcul. Cela tient peut-être à l'implémentation, même si elle semble aujourd'hui relativement performante (au moins pour les besoins académiques).

8.2 Perspectives

8.2.1 Analyse de l'influence de différentes hypothèses

Les résultats obtenus en utilisant une méthode de type Monte Carlo sont très prometteurs et permettent d'envisager de nombreuses applications.

Comme expliqué au chapitre VII, les résultats obtenus sont représentatifs à hauteur des entrées. Si la distribution en entrée de notre algotrithme est biaisé, il en sera tout autant des résulats. Cela nous permet d'envisager une première perspective à la suite de ces travaux : l'étude de l'influence de différentes hypothèses.

Par exemple, considérons la question de l'influence des hypothèses sur la loi de distribution des *offsets*. Faut–il contraindre toutes les tâches à démarrer en un temps borné et

inférieur à leur période, ou vaut-il mieux les espacer et de combien ? Donner des valeurs particulières aux *offsets* est-il intéressant, et lesquelles ? Par exemple, faut-il privilégier les multiples des périodes de la tâche la plus rapide ? Il existe de nombreuses pistes dand la littérature.

Plus généralement, l'étude de l'influence d'autres paramètres est également une perspective intéressante. Par exemple, pour un système avec une charge donnée, comment varient les temps de réponse moyens et pire cas avec les durées d'exécution des tâches et leurs périodes. Vaut-il mieux de petites tâches souvent, ou de grosses tâches plus rarement ? Où encore, pour un jeu de tâches donné, est-ce que les temps de réponse sont sensibles à de petites variations dans les périodes et/ou les durées d'exécution ? Cela permettrait par exemple de modifier légèrement la configuration du système si le gain devait être significatif.

8.2.2 Prise en compte de plusieurs priorités

Dans ces travaux, nous n'avons considéré qu'une seule priorité parmi les tâches du système. La situation se complique rapidement lorsqu'il y a plusieurs priorités à prendre en compte. En effet, nous sommes en contexte non-préemptif ; toute tâche déjà rangée dans la file d'attente peut être retardée par d'autres tâches plus prioritaires, même si elles arrivent plus tard. Par contre, dès qu'une tâche est sélectionnée pour être exécutée, elle est sûre de ne pas être interrompue par d'autres tâches. Mais comment prendre en compte ces tâches plus prioritaires ?

Nous proposons la piste d'approfondissement suivante. Sans perte de généralité et dans un souci de simplification, nous ne considérons que deux priorités. Tant qu'il n'y a pas d'instances de tâches de priorité haute, rien ne change.

Quand une tâche de priorité haute est activée, il faut déterminer les différents scénarios de *backlog* qui dépendent du rangement des instances de priorité inférieure dans la file d'attente à l'instant précédent. Nous qualifions ces instants *d'instants de branchement*. Par exemple, supposons qu'à l'instant d'activation précédent l'instant t, deux tâches de priorité basse soient dans la file d'attente et que le processeur soit libre. Le *backlog* vu par la tâche de priorité haute à l'instant t dépend du « choix » du processeur pour la prochaine tâche à exécuter à l'instant précédent. Si la tâche qu'il choisit est la plus grosse, la tâche de priorité haute devra attendre plus longtemps.

La complexité de ce problème d'identification de ce que nous appelons scénarios de *backlog* est NP-difficile, et revient à trouver parmi un ensemble d'entiers, quels sont les sous-ensembles dont la somme des éléments est donnée à l'avance. Et c'est précisément le problème que nous essayons de résoudre.

Une fois les scénarios identifiés, il faut les sauvegarder pour un traitement ultérieur et avancer dans le temps jusqu'au prochain instant de *backlog* nul. À cet instant, le passé n'a plus d'influence sur l'avenir et le problème est de nouveau le même que précédemment. À la fin de l'hyperpériode, il faut reprendre la liste des différents instants et scénarios mis de côté à l'étape précédente en faisant avancer le temps pour chacun à après l'exécution de la tâche de priorité haute.

Et ainsi de suite… L'algorithme s'arrête lorsqu'il n'y a plus d'instants de branchement à traiter. Cet algorithme, qui s'apparente à du *backtracking*, n'a pas pu être implémenté durant ces travaux par manque de temps. Cependant, s'il l'était, il est possible que sa durée d'exécution soit trop rédhibitoire pour qu'il soit utile sur des cas d'application concrets. Cela reste à vérifier cependant, avec notamment, une implémentation performante.

8.2.3 Prise en compte des tâches sporadiques et de plusieurs nœuds

Une tâche sporadique a également une période, mais sa signification est différente. La période d'une tâche sporadique est la durée *minimale* qui sépare deux activations successives. Cela signifie que la durée entre deux activations est bornée par le bas, mais pas par le haut. Comment prendre en compte ce type de tâche dans notre algorithme ? C'est très simple en fait car rien ne change sur le calcul des lois de distributions des temps de réponse à proprement parlé. Il y a deux points délicats cependant : la génération de la trace symbolique des activations et la durée de la fenêtre d'étude.

Il faut un moyen de déterminer les instants d'activation des tâches sporadiques dans la mesure où la seule connaissance des *offsets* n'est plus suffisante. C'est aussi le cas si l'on considère une chaîne de commutateurs. En effet, il faut déterminer les instants d'arrivée des trames dans un nœud en fonction du délai à la sortie du nœud précédent.

Une première solution consiste à considérer que les activations se font aussi rapidement que possible : c'est le cas strictement périodique. Dans la théorie temps réel classique initiée par les travaux de Liu & Layland, cette hypothèse combinée à la synchronicité des tâches au démarrage conduit au scénario mettant en évidence les pire temps de réponse justement. Cela pose une question fondamentale : quelle est la représentativité de résultats obtenus de cette façon par rapport à une exécution réelle et comment l'évaluer ?

D'un autre côté, si l'on dispose du modèle des activations de ces tâches sporadiques, il est possible de s'en servir pour générer une trace représentative d'une exécution réelle dans la limite de celle du modèle des activations. Encore une fois, même si les entrées sont plus représentatives d'un comportement réel du système, comment évaluer si les résultats sont conformes à la réalité et dans quelle mesure ?

D'autre part, les activations des tâches ne sont plus cycliques et cela pose le problème de la fenêtre d'étude. Le problème est le même que pour les méthodes basées sur la simulation : il faut avoir confiance dans le fait que la durée d'étude est suffisamment représentative d'une exécution réelle. Comment évaluer la confiance d'une solution sur une fenêtre finie par rapport à la solution hypothétique sur une fenêtre infinie ? Nous n'avons pas de solution à ce problème autre que de faire tourner l'algorithme « suffisamment longtemps » pour se donner une confiance qualitative.

Le lecteur attentif aura remarqué que les VL en AFDX s'apparentent à des tâches sporadiques. Mais nous avons fait l'hypothèse qu'ils étaient strictement périodiques, quitte à envoyer un trafic nul. Cela reste une hypothèse crédible dans le domaine des réseaux. Cependant, prendre en compte la variabilité des périodes augmenterait significativement la durée descalculs nécessaires pour garder une précision intéressante.

8.2.4 Prise en compte des tâches de durée variable

Dans ces travaux, nous avons supposé que les durées d'exécution des tâches étaient constantes, égale au WCET. Dans la pratique, les tâches sont des programmes informatiques pouvant comporter des boucles, des branchements conditionnels et autres constructions rendant la durée de chaque exécution potentiellement différente des autres.

Il s'agit d'une piste intéressante d'approfondissement. La difficulté majeure consiste à trouver une manière d'utiliser les motifs. En effet, considérer toutes les valeurs possibles entre o et WCET va considérablement augmenter la taille de la table des motifs et donc le besoin en temps et en espace.

Dans le cas de réseaux, critiques de surcroît où tout (ou presque) est connu de manière statique, il est plus envisageable de considérer des tailles fixes pour les trames. En effet, les données envoyées sont toujours les mêmes — température, altitude, pression, etc. — et leur codage ne change pas. Quitte à remplir quelque trame de zéros, il est possible d'imaginer que les trames d'un VL pourraient avoir toutes la même taille. De nouveau, dans un contexte de réseau cette hypothèse est crédible, mais une bonne précision nécessiterait de longs calculs.

8.2.5 Prise en compte de la dérive des horloges

Nous reprenons ici les propos du chapitre VI.

De deux choses l'une, soit la séquence de démarrage d'un système temps réel est connue à l'avance, soit ce n'est pas le cas. Dans ce cas, il n'est pas impossible que les *offsets* des tâches soient différents d'un allumage du système à l'autre.

D'autre part, soit les tâches d'un système temps réel distribué, comme un réseau, sont synchronisées sur une horloge commune ou partagée, soit leurs horloges propres dérivent les unes par rapport aux autres. En effet, les quartz de ces horloges ne sont pas strictement identiques et leur fréquences sont légèrement différentes. Cela suffit à ce que les horloges dérivent dans le temps.

Bien que différents, la conséquence de ces deux phénomènes est identique : les lois de distribution des temps de réponse calculées pour un seul jeu d'*offsets* sont insuffisantes pour caractériser le comportement des systèmes temps réel dans la durée. Il faut mener ces calculs pour plusieurs jeux d'*offsets*, et même, *beaucoup* de jeux.

Dans la pratique, il y a bien trop de combinaisons des *offsets* pour effectuer un calcul complet. Dans ces travaux, nous avons étudié la possibilité d'utiliser des méthodes de type Monte Carlo pour analyser l'influence de ce phénomène. Par ailleurs, nous avons proposé d'étudier différentes hypothèses supplémentaires sur les *offsets* comme le fait que les tâches démarrent toutes en moins de X par exemple. D'autres approches seraient également intéressantes : la littérature est riche d'exemples. Tout reste ouvert.

Annexes

Références A

[1] , *DO-254 : Design assurance guidance for airborne electronic hardware* (2000).

[2] , *DO-297 : Integrated Modular Avionics (IMA) Development Guidance and Certification Considerations* (2000).

[3] , *SAE ARP-4754A : Guidelines for Development of Civil Aircraf and Systems* (2010).

[4] , *DO-178C : Software Considerations in Airborne Systems and Equipment Certification* (2012).

[5] T. Amnell *et al.*, UPPAAL : now, next, and future, in *Modeling and verification of parallel processes*, edited by F. Cassez, C. Jard, B. Rozoy, and M. D. Ryan , number 2067 in Lecture Notes In Computer Science (Springer-Verlag New York, Inc.), p. 99–124.

[6] N. Audsley, *Optimal Priority Assignment And Feasibility Of Static Priority Tasks With Arbitrary Start Times* (York, England, 1991).

[7] P. Barbe and M. Ledoux, *Probabilité*. (EDP Sciences, 2007).

[8] S. K. Baruah, N. K. Cohen, C. G. Plaxton, and D. A. Varvel, *Proportionate Progress : A Notion of Fairness in Resource Allocation*, Algorithmica **15** (1996), no. 6, 600–625

[9] H. Bauer, J.-L. Scharbarg, and C. Fraboul *Applying and optimizing trajectory approach for performance evaluation of AFDX avionics network*, in *Emerging Technologies and Factory Automation (ETFA)* (IEEE, 2009b).

[10] H. Bauer, J.-L. Scharbarg, and C. Fraboul *Applying trajectory approach to AFDX avionics network*, in *ECRTS (session WiP)* (Dublin, 2009a).

[11] M. Boyer and C. Fraboul *Tightening end to end delay upper bound for AFDX network with rate latency FCFS servers using network calculus*, in *Proceedings of the 7th IEEE Int. Workshop on Factory Communication Systems Communication in Automation (WFCS 2008)* (IEEE industrial Electrony Society, 2008), pp. 11–20.

[12] M. Boyer, L. Jouhet, and A. Bouillard *Notations pour le calcul réseau*, in *Modélisation des systèmes réactifs (MSR)* (JESA, 2009).

[13] F. Carcenac, Ph.D. thesis, École Nationale Supérieure de l'Aéronautique et de l'Espace, (2005).

[14] F. Carcenac and F. Boniol, *A formal framework for verifying distributed embedded systems based on abstraction methods*, J-Int-J-Softw-Tools-Technol-Transfer **8** (2006), no. 6, 471–484

[15] F. Carcenac, F. Boniol, and Z. Mammeri *Verification of an Avionic System Using Timed Model Checking*, in *First International Symposium on Leveraging Applications of Formal Methods, ISoLA* (Paphos, Cyprus, 2004).

[16] H. Charara, J.-L. Scharbarg, J. Ermont, and C. Fraboul *Methods for bounding end-to-end delays on an AFDX network*, in *Real-Time Systems, 2006. 18th Euromicro Conference on* (2006a).

[17] T. Bonald, A. Proutiere, and J. Roberts *Statistical performance guarantees for streaming flows using expedited forwarding*, in *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (2001), pp. 1104–1112 vol.2.

[18] T. F. Chan, G. H. Golub, and R. J. Leveque, *Algorithms for Computing the Sample Variance : Analysis and Recommendations*, The American Statistician **37** (1983), no. 3, 242–247

[19] H. Charara, Ph.D. thesis, Institut National Polytechnique de Toulouse, (2007).

[20] H. Charara, J.-L. Scharbarg, and C. Fraboul *Focusing simulation for end-to-end delays analysis on a switched Ethernet*, in *RTSS 2006 - Work in Progress session, Rio de Janeiro, 05/12/2006-08/12/2006* (IEEE, http ://www.ieee.org/, 2006b), pp. 65–68.

[21] A. Choquet-Geniet and E. Grolleau, *Minimal schedulability interval for real-time systems of periodic tasks with offsets*, Theor. Comput. Sci. **310** (2004), no. 1-3, 117–134

[22] R. Cruz, *A calculus for network delay. I. Network elements in isolation*, IEEE Transactions on Information Theory **37** (1991a), no. 1, 114–131

[23] R. Cruz, *A calculus for network delay. II. Network analysis*, IEEE Transactions on Information Theory **37** (1991b), no. 1, 132–141

[24] R. I. Davis and A. Burns, *A survey of hard real-time scheduling for multiprocessor systems*, ACM Comput. Surv. **43** (2011), no. 4, 35

[25] G. Dufour and G. Durrieu, *Modélisation fluide de réseaux* Tech. Rep., Onera (Office National d'Étude et de Recherche Aérospatiale) (2009).

[26] A. K. Erlang *The Theory of Probabilities and Telephone Conversations*, in *Nyt Tidsskrift for Matematik* (1909), pp. 33–39.

[27] A. K. Erlang *Solution of some Problems in the Theory of Probabilities of Significance in Automatic Telephone Exchanges*, in *Electroteknikeren* (1917), pp. 5–13.

[28] J. Goossens and L. Cucu-Grojean, *Scheduling of Offset Free Systems*, Real-Time Systems **24** (2003), no. 2, 239–258

[29] G. Gopalakrishnan and S. Qadeer, eds., *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, number 6806 in Lecture Notes in Computer Science (Springer, 2011).

[30] J. Grieu, Ph.D. thesis, Institut National Polytechnique de Toulouse, Toulouse, France, (2004).

[31] K. Jeffay and D. F. S. C. U. Martel *On non-preemptive scheduling of periodic and sporadic tasks*, in *Proceedings of 12th IEEE Real-Time Systems Symposium* (1991), pp. 129–139.

[32] M. Joseph and P. K. Pandya, *Finding Response Times in a Real-Time System*, Comput. J. **29** (1986), no. 5, 390–395

[33] N. Jun, H. Ip, and S. A. Edwards *A processor extension for cycle-accurate real-time software*, in *Embedded and Ubiquitous Computing, volume 4096 of LNCS* (Springer, 2006), pp. 449–458.

[34] L. Kleinrock, *Computer Applications, Volume 2, Queueing Systems* (Wiley-Interscience, 1976).

[35] D. E. Knuth, *The Art of Computer Programming, Volume II : Seminumerical Algorithms, 2nd Edition.* (Addison-Wesley, 1981).

[36] M. Z. Kwiatkowska, G. Norman, and D. Parker *PRISM 4.0 : Verification of Probabilistic Real-Time Systems* In [29], pp. 585–591.

[37] M. Lauer, Thèse de doctorat, Institut National Polytechnique de Toulouse, Toulouse, France, (2012).

[38] J.-Y. Le Boudec and P. Thiran, *Network Calculus : A Theory of Deterministic Queing System for the Internet.* (Springer Verlag, 2004).

[39] E. A. Lee, *Computing Needs Time*, Communications of the ACM **52** (2009), no. 5, 70–79

[40] J. P. Lehoczky *Fixed Priority Scheduling of Periodic Task Sets with Arbitrary Deadlines*, in *IEEE Real-Time Systems Symposium* (1990), pp. 201–213.

[41] J. Y.-T. Leung and M. L. Merrill, *A Note on Preemptive Scheduling of Periodic, Real-Time Tasks*, Inf. Process. Lett. **11** (1980), no. 3, 115–118

[42] J. Y. T. Leung and J. Whitehead, *On the complexity of fixed-priority scheduling of periodic, real-time tasks*, Performance Evaluation **2** (1982), no. 4, 237–250

[43] B. Lickly *et al. Predictable programming on a precision timed architecture*, in *Proceedings of the International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)* (ACM, 2008), pp. 137–146.

[44] R. F. Ling, *Comparison of Several Algorithms for Computing Sample Means and Variances*, Journal of the American Statistical Association **69** (1974), no. 348, 859–866

[45] C. Liu and J. Layland, *Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment*, J. ACM **20** (1973), no. 1, 46–61

[46] C. L. Liu, *Scheduling Algorithms for Multiprocessors in a Hard Real-Time Environment*, JPL Space Programs Summary 37-60 **II** (1969), 28–31

[47] J. Mäki-turja and M. Nolin *Tighter Response-Times for Tasks with Offsets*, in *In Proc. of the 10 th International conference on Real-Time Computing Systems and Applications (RTCSA'04)* (IEEE, 2004).

[48] S. Martin, Ph.D. thesis, Université Paris XII, (2004).

[49] C. Mauclair, *Analysis of Real-Time Networks with Monte Carlo Methods*, EUCASS'11 FLIGHT DYNAMICS/GN&C and AVIONICS for Aeronautic and Space Applications **6** (2011a)

[50] C. Mauclair, *Procédé de calcul de la distribution de la durée de traversée d'un commutateur de communication multiplexé* (2011b). Patent number 11 01935.

[51] K. L. McMillan, Ph.D. thesis, Carnegie Mellon University, (1993).

[52] P. Minet, S. Martin, L. Azouz Saidane, and S. Azzaz *FP/FIFO scheduling : coexistence of deterministic and probabilistic QoS guarantees*, in *DMTCS* (2007).

[53] A. K. Mok, Ph.D. thesis, Massachussets Institute of Technology, Massachussets USA, (1983).

[54] P. Moreira *et al. White rabbit : Sub-nanosecond timing distribution over Ethernet*, in *Precision Clock Synchronization for Measurement, Control and Communication, ISPCS* (2009), pp. 1–5.

[55] J. C. Palencia, J. J. García Gutiérrez, and M. González Harbour *On the Schedulability Analysis for Distributed Hard Real-Time Systems*, in *Proceedings of IEEE Euromicro Real-time systems* (1997).

[56] F. Ridouard, J.-L. Scharbarg, and C. Fraboul *Stochastic network calculus for buffer overflow evaluation in an avionics switched Ethernet*, in *Junior Researcher Workshop on Real-Time Computing 2007 JRWRTC'07* (2007b).

[57] F. Ridouard, J.-L. Scharbarg, and C. Fraboul *Stochastic network calculus for end-to-end delays distribution evaluation on an avionics switched Ethernet*, in *Industrial Informatics, 2007 5th IEEE International Conference* (2007a), pp. 559–564.

[58] F. Ridouard, J.-L. Scharbarg, and C. Fraboul *Stochastic upper bounds for heterogeneous flows using a Static Priority Queueing on an AFDX network*, in *Journées FAC'2008, SVF/FéRIA* (2008).

[59] J. Serrano *et al.*, *The White Rabbit Project* Tech. Rep. CERN-ATS-2009-096, CERN, Geneva (2009).

[60] L. Sha, R. Rajkumar, and J. P. Lehoczky, *Priority Inheritance Protocols : An Approach to Real-Time Synchronization*, IEEE Trans. Computers **39** (1990), no. 9, 1175–1185

[61] C. R. Spitzer, *The Avionics Handbook*. (CRC Press, 2000).

[62] I. Stoica *et al. A proportional share resource allocation algorithm for real-time, time-shared systems*, in *IEEE Real-Time Systems Symposium* (1996), pp. 288–299.

[63] J. Sun, Ph.D. thesis, University of Illinois, (1997).

[64] K. Tindell, *Using Offset Information to Analyse Static Priority Pre-emptively Scheduled Task Sets* Tech. Rep. YCS-182, Dept. of Computer Science, University of York, England (1992).

[65] K. Tindell and J. Clark, *Holistic Schedulability Analysis for Distributed Hard Real-Time Systems*, Microprocess. Microprogram. **40** (1994), no. 2–3, 117–134

[66] M. Vojnović and J.-Y. Le Boudec *Stochastic Analysis of Some Expedited Forwarding Networks*, in *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE* (2002b), pp. 1004–1013.

[67] B. P. Welford, *Note on a Method for Calculating Corrected Sums of Squares and Products*, Technometrics **4** (1962), no. 3, 419–420

Prototype développé B

B.1 Choix du langage

Le langage utilisé pour le développement du prototype devait répondre aux deux besoins essentiels que sont des performances élevées et une bonne connaissance du langage par l'auteur. Pour répondre à la problématique de performance, le choix s'est rapidement fixé sur un langage compilé. La prise en compte de l'autre contrainte a réduit les possibilité aux langages C et Fortran 90. Pour le choix final, nous avons tenu compte des compétences de l'équipe et du département qui compte de nombreux mathématiciens rompus au langage Fortran. Par ailleurs, la souplesse des notations tableaux, les fonctions intrinsèques sur ces derniers et une gestion des pointeurs qui paraissait plus simple ont fini de confirmer ce choix.

À refaire ce choix aujourd'hui, le langage interprété Lua serait préféré pour sa simplicité, notamment de sa structure de données unique et sa gestion automatique de la mémoire. D'autre part, la logique initiale de calcul basée sur des tableaux a peu à peu été revue et l'apport de Fortran sur ces aspects n'est donc plus un avantage. Pour des besoins éventuels d'amélioration des performances du prototype, il serait toujours possible d'implémenter certains morceaux en C et de réaliser une interface (un processus très simple en Lua).

B.2 Logique implémentée

Le prototype est passé par une phase préliminaire et l'utilisation intensive de tableaux afin d'obtenir un code compact et compréhensible. Néanmoins après quelque temps, il a fallu nous rendre à l'évidence que cet objectif n'était pas atteint, en particulier lorsqu'il a fallu retourner modifier le code plusieurs mois après. Ce dernier a donc été réécrit en essayant de corriger certaines erreurs et en prenant quelques hypothèses simplificatrices. Dans sa version actuelle, chaque fonction et structure de donnée à un commentaire court sur sa raison d'être et le total cumulé est inférieur à 1000 lignes de code.

Les grandes étapes de l'algorithme sont les suivantes.

1. Lecture de la configuration à partir d'un fichier d'entrée qui doit respecter le format suivant : nombre de tâches du système, puis par tâche, C_i, T_i et r_i, une par ligne. Les C_i doivent être triés, dans l'ordre croissant ou décroissant. Le nombre de tâches a été introduit pour simplifier la lecture en Fortran. Le fichier de test utilisé au cours du développement représente le cas synchrone :

```
8
10 100 0
10 200 0
10 200 0
10 200 0
4   60 0
4   30 0
5   30 0
1    8 0
```

2. Détermination des classes et de la tables des motifs. L'algorithme de détermination des classes est très simple et suppose qu'elles sont triées en comparant la valeur courante avec la précédente, si la valeur change, alors il y a une nouvelle classe. Pendant un temps, un nouvel algorithme a été envisagé : la génération des motifs par taille constante. Cette solution a été implémentée très simplement en C avec des implémentations récursive et itérative (fournie avec le prototype). Néanmoins, la recherche n'a pas semblé plus simple ou rapide et une approche directe a finalement été retenue. Chaque colonne comprend chaque valeur de o au cardinal de la classe un certain nombre de fois, organisées en blocs. La première colonne contient un seul bloc où chaque valeur est répétée $\prod_{m=2}^{k} \left(\operatorname{card}\left(\mathbb{C}_m\right)+1\right)$ fois. Puis, la deuxième colonne se compose de $\left(\operatorname{card}\left(\mathbb{C}_1\right)+1\right)$ blocs, où chaque valeur se répète $\prod_{m=3}^{k} \left(\operatorname{card}\left(\mathbb{C}_m\right)+1\right)$ fois… La p$^{\text{e}}$ colonne se compose de $\prod_{m=1}^{p-1} \left(\operatorname{card}\left(\mathbb{C}_m\right)+1\right)$ blocs, où chaque valeur se répète $\prod_{m=p+1}^{k} \left(\operatorname{card}\left(\mathbb{C}_m\right)+1\right)$ fois… La dernière colonne change de valeur à chaque ligne. Cet algorithme génère une table des motifs dans le même style que la représentation des entiers naturels en base 2.

```
!> Repetitions of a pattern
!!   - 'value': times each value in block will be repeated in block
!!   - 'block': times each block of values will be repeated in column
SUBROUTINE pattern_repetitions(pattern, blocks, values)
  ! parameters descriptions
  INTEGER, DIMENSION(:), POINTER :: pattern, blocks, values
  ! internal variables
  INTEGER :: i
  blocks(1) = 1
  values(1) = PRODUCT(pattern+1) / (pattern(1)+1)
  DO i = 2, UBOUND(pattern, 1)
     blocks(i) = blocks(i-1) * (pattern(i-1)+1)
     values(i) = values(i-1) / (pattern(i)  +1)
  ENDDO
ENDSUBROUTINE pattern_repetitions
```

3. Prise en compte des *offsets* des tâches pour le calcul de la trace symbolique des activations avec les motifs activés. La méthode utilisée est très simple également. Pour tous les temps d'activation de chaque tâche sur l'intervalle d'étude $[0, \max r_i + 2 \cdot \operatorname{ppcm} T_i[$,

la tâche est ajoutée au motif de l'instant correspondant. L'utilisation d'un arbre facilite la création de la trace au fur et à mesure.

4. Calcul de la distribution pour chaque tâche sur un intervalle semi-ouvert paramétrable (entre $\max r_i + \text{ppcm}\, T_i$ et $\max r_i + 2 \cdot \text{ppcm}\, T_i$ dans nos exemples, mais un autre serait tout à fait possible). Pour chaque tâche et à chacun de ses temps d'activation, le motif maître est déterminé pour la tâche, ses motifs esclaves calculés et le délai pondéré de chacun est alors ajouté à la distribution. La pondération prend en compte le poid relatif de l'activation parmi toutes les activations dans l'intervalle. Cela revient à normaliser la distribution d'une instance par le nombre d'activation de la tâche sur l'hyperpériode (voir les explications à la section 6.5).

B.3 Automatisation et post-traitements

Le prototype développé ne gère qu'un jeu d'*offsets*. Nous avons donc développé un script Lua pour automatiser une procédure de test. Le script génère un nombre donné de jeux d'*offsets* différents et pseudo-aléatoires (la graîne du générateur peut être renseignée pour des besoins de tests) et appelle à chaque fois le prototype avec le bon format des paramètres. Dans une première version, un fichier de configuration était créé à chaque fois et chaque fichier de résultat était également sauvegardé. Le nombre de fichiers à conserver devenant grand (plusieurs millions), nous avons préféré ne plus les sauvegarder. Une option permet de forcer « l'apparition » du cas synchrone parmi les jeux d'*offsets*à des fin de test.

```
Compute different results of a system of real-time tasks

  -r, --results      (string default 'all') Type of results to print (all, full, local, mini, maxi)
  -t, --tasks        (string)               Description of the tasks: '(Ci, Ti) (Ci, Ti)...'
  -n, --offsets      (number default 1)     Number of offsets to generate
  -s, --synchronized                        Force the synchronous offsets scenario
      --seed         (number default 0)     Use given seed for repeatable computations
                                            (0 means random)
  -p, --pretty                              Pretty print (only for 'full')
```

Le paramètre `results` permet de filtrer la sortie du programme pour ne retenir que certains extraits.

Un autre script permet ensuite de calculer les différents paramètres statistiques présentés au chapitre VII. Il permet également de normaliser les résultats.

```
Launch some computations

  -n                 (number)              Number of tasks (to ease parsing)
  -v, --verbose                            Print information about progress
  -f, --filter                             Print the input as is (use program as filter)
  -t, --total                              Print the totals
  -s, --stats                              Print some statistics
  -d, --distribution                       Print the distributions
  -r, --transposed                         Print results transposed, without header
      --sep          (string default " ")  Separator
  -o, --output       (string default "")   Output file ("" means stdout)
  <input_file>       (default stdin)       File to be normalized
```

B.4 Remarques

Tests

L'ensemble prototype plus scripts a été testé de nombreuses fois sur des cas simples pour s'assurer de la correction des algorithmes implémentés. Cependant, aucun test unitaire n'a été écrit, ce que nous reconnaissons volontier être une situation qu'il faudrait améliorer.

Performances

Sur l'exemple support (8 tâches, 4 classes, hyperpériode de 600), le temps de calcul du prototype est de l'ordre de 0.15ms : 10^6 configurations en 2h30. C'est encourageant, mais la génération de l'ensemble de toutes les configurations possibles des *offsets* prendrait tout de même un peu plus de quatre ans et neuf mois sur un système monoprocesseur. Cela pourrait être amélioré si la construction de la table des motifs n'était pas effectué à chaque fois, mais une seule fois au début. Pour le prototype cependant, nous n'avons pas ressenti le besoin d'améliorer ce point.

Un intérêt non négligeable de notre approche de type Monte Carlo est que nous pouvons tout à fait utiliser plusieurs cœurs et/ou plusieurs ordinateurs pour effectuer des calculs de manière parallèle. En effet, les calculs sont indépendants entre eux. Il faut ensuite simplement réunir les différents fichiers résultats avant de normaliser les résultats. Par exemple, dix ordinateurs pourraient être utilisés pour passer à un échantillon de 10^7 en « 2h30 seulement ».

Une approche statistique des réseaux temps réels embarqués

Depuis quelques années, les réseaux de communication déployés au sein d'aéronefs sont toujours plus vastes et plus complexes. Ces bus numériques multiplexent différents flux de données afin de limiter les câbles, mais cela induit des retards sur les transmissions.
Les travaux présentés ici portent sur une approche statistique de l'évaluation des performances du pire temps de traversée d'un réseau embarqué de type AFDX. Il s'agit de définir une nouvelle approche visant à associer à un calcul pire cas, une distribution des temps de transmission des messages, en vue notamment de permettre d'apprécier le pessimisme du calcul pire cas. Les méthodes décrites sont applicables dans le cadre plus général d'un ensemble de tâches.
Nous proposons trois contributions dans ces travaux. Tout d'abord, une méthode originale d'évaluation de la distribution de la durée de traversée d'un commutateur AFDX qui s'appuie sur une énumération symbolique des scénarios d'ordonnancement dans la file d'attente.
Puis, un algorithme efficace de calcul des délais subis par des messages/tâches périodiques lorsque les déphasages initiaux sont connus. Les délais calculés sont exacts ainsi que la distribution de probabilité.
Enfin, le calcul de la distribution des délais subis par des messages/tâches dans un cadre général, à l'aide d'une méthode statistique de type Monte Carlo. Des décalages initiaux sont tirés aléatoirement et permettent de nourrir l'algorithme précédent.

Mots–clés : temps réel, réseaux embarqués, AFDX, Monte Carlo, énumération symbolique

A statistical approach to embedded real-time networks

Since a few years, communication networks deployed in aircrafts are ever larger and ever more complex. These digital buses multiplex different data streams in order to save cabling, but this causes delays on transmissions.
The work presented here is based on a statistical evaluation of the worst case transit time of an embedded network of the AFDX type. It consists in associating a worst case computation with a complete distribution of the transit times in order, among other things, to appreciate the pessimism of worst case approaches. The methods are also applicable to a set of real-time tasks.
This work contributes three major results. First, an original method to evaluate the distribution of the transit time through an AFDX switch, based on the symbolic enumeration of the scheduling scenarios in the waiting queues of the switch.
Second, an effective algorithm to compute the delays encountered by periodic messages/tasks when initial offsets are known. Delays thus computed are exact and so is the delays distribution.
Third, the computation of the delays distribution encountered by messages/tasks in a general case using a Monte Carlo based statistical method. Initial offsets are randomised and feed the preceding algorithm.

Keywords : real-time, embedded networks, AFDX, Monte Carlo, symbolic enumeration