Analyse des temps de réponse et de la demande processeur en ordonnancement temps réel de tâches périodiques

Pascal Richard
Laboratoire d'Informatique Scientifique et Industrielle
ENSMA et Université de Poitiers

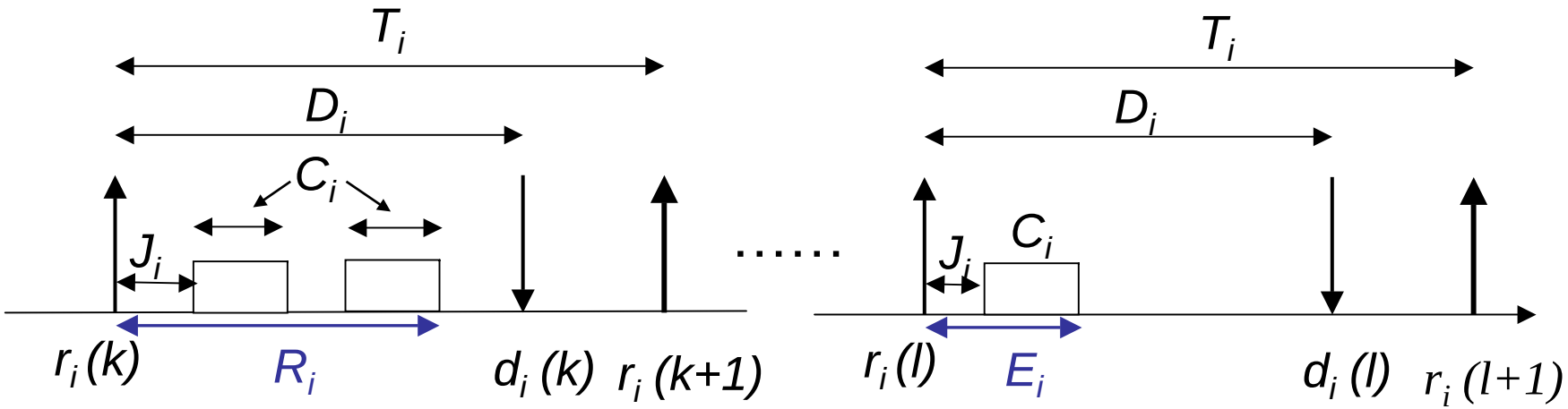
pascal.richard@univ-poitiers.fr

Sommaire

1. Introduction
2. Principes des deux Analyses
3. Tests Exacts
4. Tests Approchés
5. Les principaux problèmes rencontrés

1.1 Tâches périodiques

Activation périodique des tâches et notations :



R_i : Pire temps de réponse

E_i : Meilleur temps de réponse

1.1 Les problèmes d'ordonnancement

- Ordonnancement en-ligne :

• Ordonnancement : choix ou conception d'un l'algorithme choix du système d'exploitation temps réel (Contraintes techniques)
• Validation : vérifier le respect des contraintes temporelles Tests d'ordonnançabilité ou de faisabilité (Problèmes d'évaluation des performances)

1.2 Techniques de validation

- Analyse fondée sur le facteur d'utilisation $\left(\sum \frac{C_i}{T_i}\right)$

- Analyse de la demande processeur
- Analyse du temps de réponse

Point commun : une échéance d'une tâche n'est dépassée que dans un intervalle de temps où le processeur est actif.

1.2. Techniques d'analyse

Tâches périodiques ⟹ ordonnancement infini

Solution :

- Détermination du pire scénario d'activation des tâches pouvant engendrer le non respect d'une contrainte temporelle

- Calculer la durée cumulée d'exécution des tâches dans un intervalle d'étude fini correspondant au scénario pire cas

1.2 Techniques d'analyse

- Test est exact si :
  - Scénario d'activation (c.f. exposé précédent) : se produit toujours dans la vie du système

  Et

  - Evaluation de la demande processeur : calcul exact des durées cumulées d'exécution

1.2 Techniques d'analyse

- Test est approché si
  - Le scénario ne se produit pas nécessairement

  Ou

  - Une borne supérieure des durées cumulées d'exec. est calculée

⇨ Surdimensionnement du système temps réel

2. Principes des deux analyses

- 2.1 Évaluer l'activité du processeur
- 2.2 Analyse du temps de réponse
- 2.3 Analyse de la demande processeur

2.1 Évaluer l'activité du processeur

- La demande processeur est la durée cumulée d'exécution des tâches :

  – réveillées dans l'intervalle de temps [0,t)
    - Notations : *rbf(t)* (request bound function) ou G(t)
    - Utilisation : analyse des systèmes à priorité fixe

  – réveillées et à terminer dans l'intervalle [0,t]
    - Notations : *dbf(t)* (demand bound function) ou H(t)
    - Utilisation : analyse EDF

2.1 Fonction rbf(*i*,*t*)

Rbf(*i*,*t*) : Request Bound Function

Pour chaque tâche *i* , la durée cumulée d'exécution des tâches dans l'intervalle [0,t) consiste à:

1. Trouver le nombre *k* de requêtes de la tâche dans l'intervalle

2. Multiplier *k* par la pire durée d'exécution des tâches C_i

2.1 Fonction rbf(*i*,*t*)

- Requête *k* réveillée avant *t*

$$r_i + (k-1)T_i < t \quad \Rightarrow \quad k < \frac{t - r_i}{T_i} + 1$$

- Requête *(k+1)* réveillée après ou à la date *t*

$$r_i + kT_i \geq t \quad \Rightarrow \quad k \geq \frac{t - r_i}{T_i}$$

$$rbf(\tau_i, t) = \max\left(0, \left\lceil \frac{t - r_i}{T_i} \right\rceil\right) \times C_i$$

2.1 Fonction rbf(*i*,*t*)

- Fonction de travail du processeur sur [0,t[

$$W(t) = rbf(t) = \sum_{j=1}^{n} rbf(\tau_j, t)$$

- Fonction de travail des tâches de priorités supérieures ou égales à *i.*

$$W_i(t) = rbf_i(t) = \sum_{j=1}^{i} rbf(\tau_j, t)$$

2.1 Fonction dbf(*t*)

- Dbf(*0*,*t*) (demand bound function) :

Requêtes réveillées et à terminer dans l'intervalle [*0*,*t*]

$$dbf(0,t) = \sum_{i=1}^{n} \max\left(0, \left\lfloor \frac{t - r_i - D_i}{T_i} \right\rfloor + 1\right) \times C_i$$

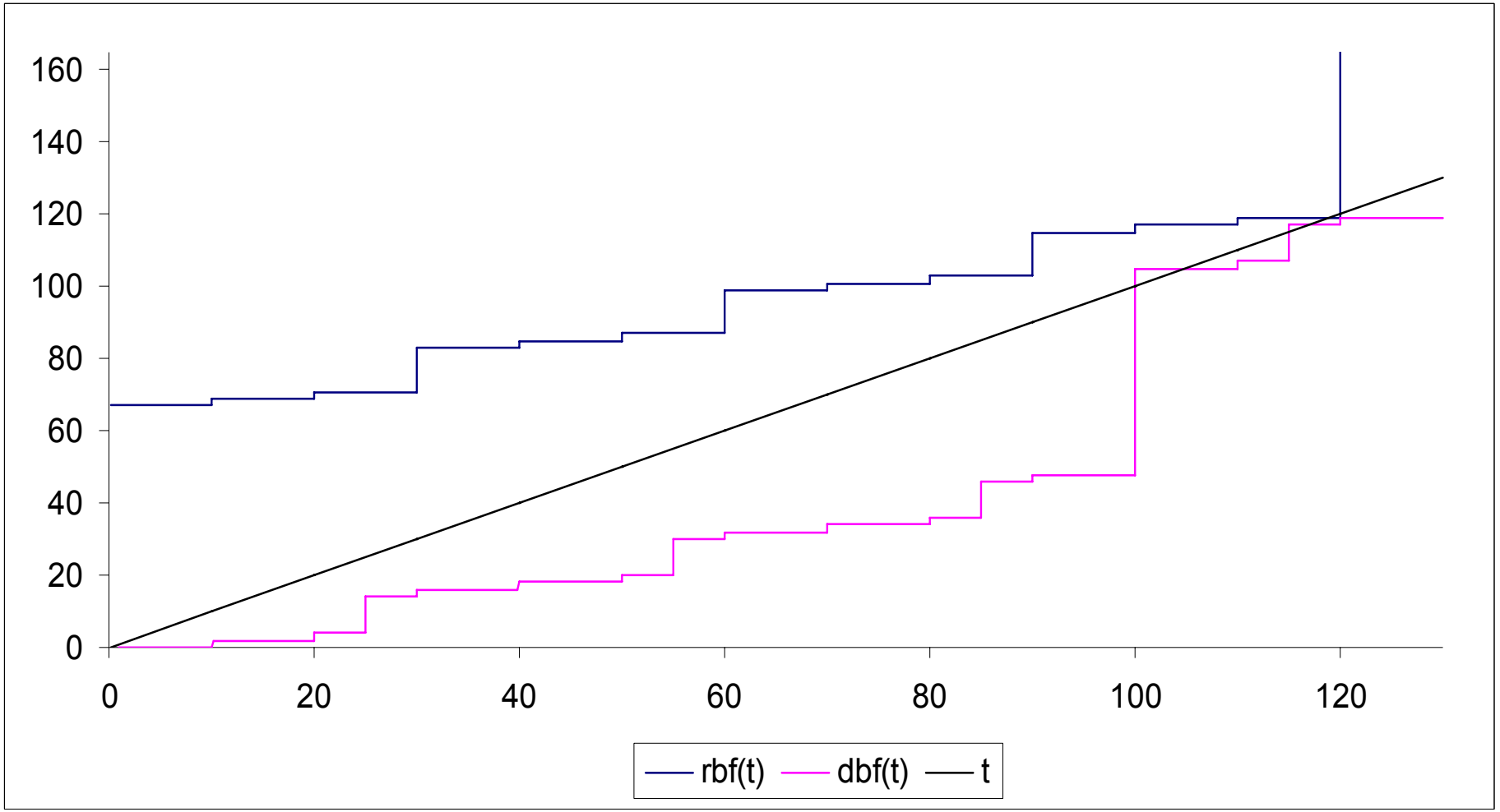
2.1 Exemple (1/2)

- Configuration de tâches (r_i=0):

Tâches	C_i	D_i	T_i
1	2	10	10
2	10	25	30
3	55	100	120

$$H = 120$$

2.1 Exemple (2/2)



2.2 Analyse du temps réponse

- Analyse tâche par tâche :
  1. Déterminer le pire scénario d'activation des tâches conduisant la tâche analysée à son pire temps de réponse

     ⇨ Etablir un résultat analytique

  2. Calculer le pire temps de réponse dans l'intervalle d'étude associé au scénario par approximation successive (formule itérative)

     ⇨ Trouver un algorithme avec une convergence rapide

  3. Vérifier $R_i \leq D_i$

2.3 Analyse de la demande processeur

- Analyse simultanée de toutes les tâches :

1. Calculer la durée cumulée d'exécution *H* des tâches dans <u>tout</u> intervalle de temps [t_1,t_2]

$$\forall t_1 < t_2 \qquad dbf(t_1, t_2) \leq t_2 - t_1$$

Rq: La notion de scénario permet de limiter le nombre d'intervalles [t_1,t_2] à étudier

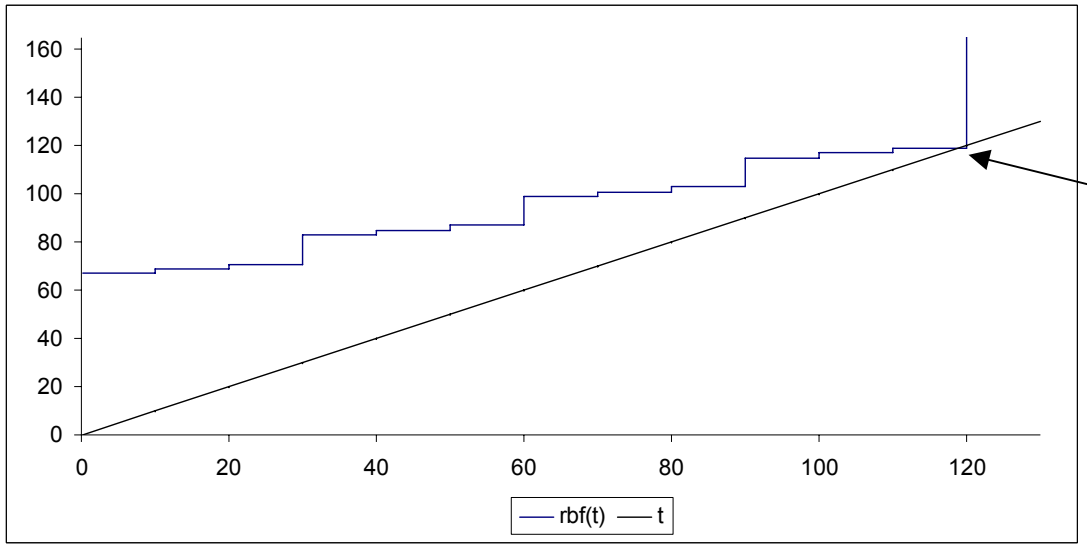
3. Tests exacts

- Analyse du temps de réponse
  - Priorité fixe
  - EDF
- Analyse de la demande processeur
  - Priorité fixe
  - EDF

3.1. Analyse temps réponse (FPP)

- Systèmes à priorité fixe (Joseph et al. 1986)

  – Résolution $W_i(t)=t$ par approximation successive (en partant d'une borne inférieure de R_i)



  – Algorithme pseudo-polynomial (Complexité : Pb ouvert)

3.1. Analyse temps réponse (EDF)

- Systèmes sous EDF
  (attention au pire scénario!)

a) Tâches périodiques (sans gigue) :

  – Aucun algorithme (pseudo-polynomial) exact connu
  – Statut Complexité : ouvert

b) Tâches sporadiques (ou périodiques avec gigue sur activation):

  – Spuri (1996) propose un algorithme exact pseudo-polynomial

3.2 Demande Processeur (FPP)

- Tâches à priorité fixe (Lehoczky, Sha et Ding 89):
  - Scénario : rechercher une faute temporelle dans [0,D_i)
  - Principe du test : Il existe une date *t* telle que la charge de travail des tâches de priorité supérieure ou égale à *i* est terminée dans l'intervalle [0,D_i)

$$\exists t \in [0, D_i) \qquad W_i(t) \leq t$$

3.2 Demande Processeur (FPP)

- Amélioration du test :
  - $W_i(t)/t$ ne change de valeur qu'à chaque nouveau réveil de tâches

$$S_i = \left\{ bT_j \,\middle|\, j = 1 \ldots i, b = 1 \ldots \left\lfloor \frac{D_i}{T_j} \right\rfloor \right\}$$

Scheduling Points ou Testing Set

  - Nombre de dates à tester dépend des paramètres des tâches (Algorithme pseudo-polynomial)

3.2 Demande Processeur (FPP)

- Formulation synthétique (et classique) du test :

$$\max_{i=1\ldots n}\left\{\min_{t\in S_i}\left(\frac{W_i(t)}{t}\right)\right\}\leq 1$$

3.2 Demande Processeur (FPP)

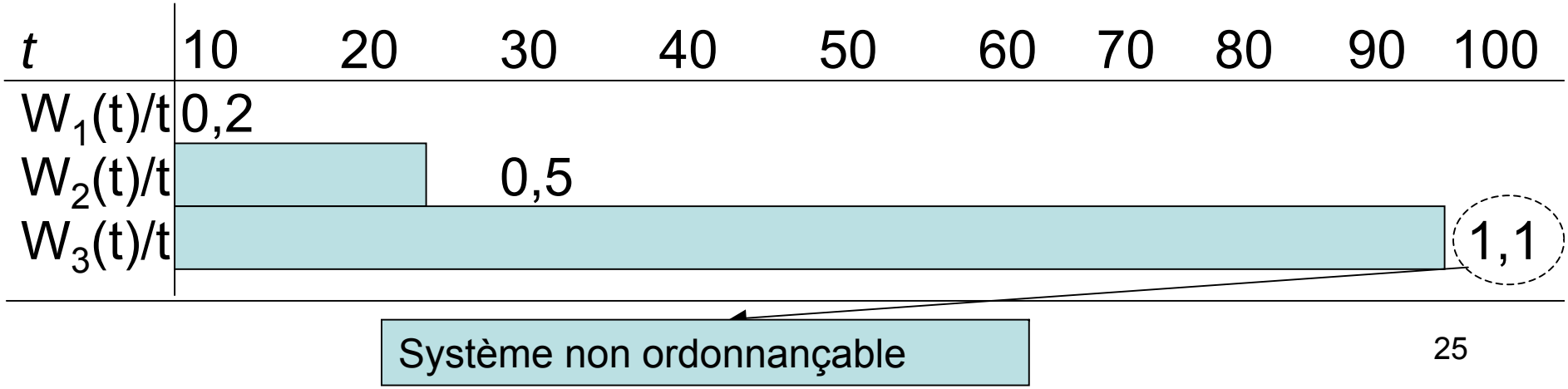
Tâches	C_i	D_i	T_i
1	2	10	10
2	10	25	30
3	55	100	120

- Exemple :

$$S_1 = \{10\}$$

$$S_2 = \{10,20,30\}$$

$$S_3 = \{10,20,30,40,50,60,70,80,90,100\}$$



3.2 Demande Processeur (FPP)

- Remarques de mise en œuvre:
  - Les tâches peuvent être analysées séparément
  - Dès qu'une date $t \in S_i$ respecte la condition alors la tâche *i* est ordonnançable
  - Ordre d'étude des dates de S_i est quelconque (plus performant de commencer par la fin)

3.2 Demande Processeur (EDF)

- Tâches périodiques à échéance sur requête

$$dbf(0,t) = \sum_{i=1}^{n} \left\lfloor \frac{t + T_i - D_i}{T_i} \right\rfloor C_i$$

- Scénario : $dbf(0, t_2 - t_1) \geq dbf(t_1, t_2) \qquad \forall t_1 < t_2$

  – La période d'étude se limite à l'intervalle [0, t_{limite}]

3.2 Demande Processeur (EDF)

- Test de *Baruah, Howell, Rosier* (1990): ordonnançable si, et seulement si,

$$dbf(0,t) \leq t \qquad \forall t,\ \ 0 < t < t_{\lim}$$

- Rq: Si pour une date *t* la condition n'est pas vérifiée, alors le système est non ordonnançable

- Il existe des améliorations limitant le nombre de points d'ordonnancement (dates *t*) à tester.

3.2 Demande Processeur (EDF)

- Exemple :

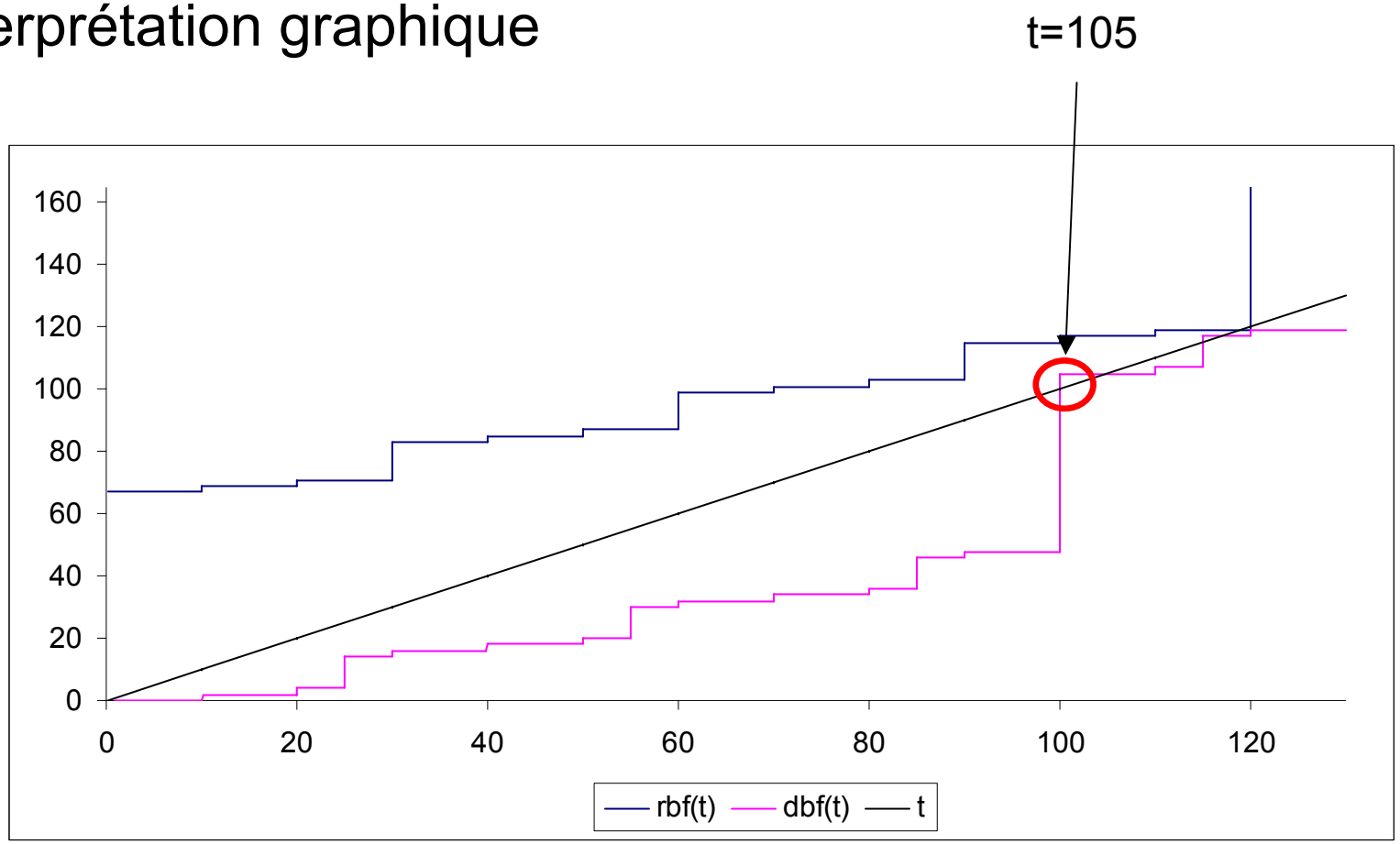
Tâches	C_i	D_i	T_i
1	2	10	10
2	10	25	30
3	55	100	120

Condition non vérifiée pour t=100

$$dbf(0,100) = 105$$

3.2 Demande Processeur (EDF)

Interprétation graphique



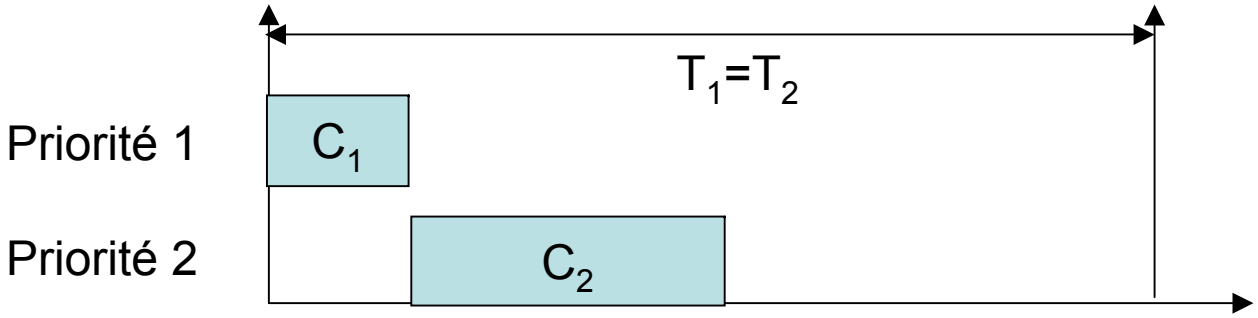
4. Tests Approchés

- Intérêts :
  - Temps de calcul trop long
  - Admission en-ligne de tâches périodiques

- Condition suffisante d'ordonnançabilité
  - Si test répond OK $\rightarrow$ système ordonnançable
  - Sinon, pas de conclusion

4.1 Tests Approchés sans garantie

- Mais quelles sont les garanties ?
- Ex: Analyse du temps de réponse en Ordonnancement non préemptif, FPP



Temps de réponse exacts sont : $R_1^* = C_1$

$R_2^* = C_1 + C_2$

4.1 Tests Approchés sans garantie

- Analyse classique du temps de réponse :
  - Calcul en tenant compte de la plus longue tâche moins prioritaire

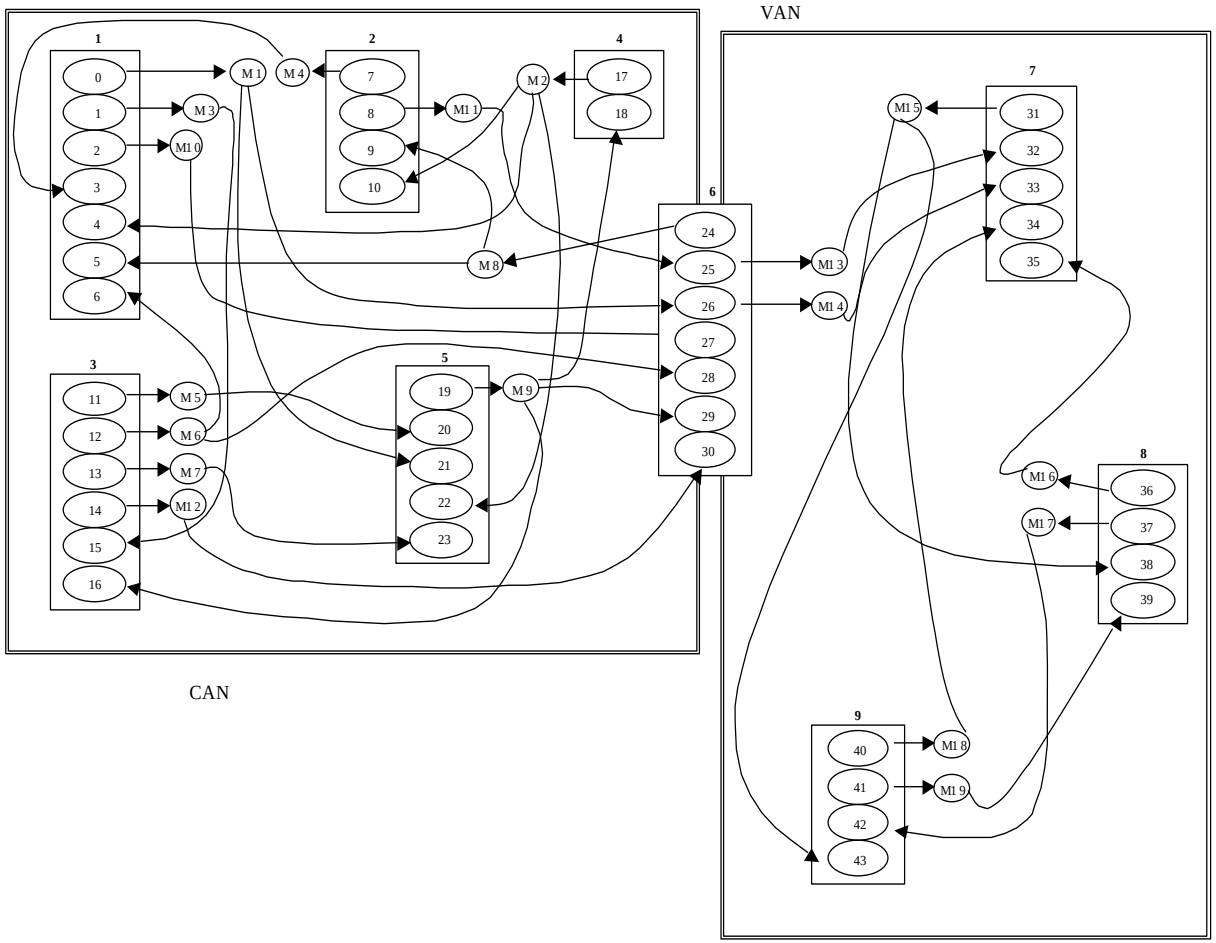
$$R_1 = C_1 + C_2$$

  - Aucune garantie de performance :

$$R_1 = \left(1 + \frac{C_2}{C_1}\right) R_1^*$$

4.1 Tests Approchés sans garantie

Analyse holistique d'un système temps réel
(analyse du temps de réponse)



4.1 Tests Approchés sans garantie

- Thèse de Ken Tindell, 1994 (York University)

Les gigues sur activation modélisent la dépendance entre les calculs des temps de réponse des tâches et des messages

Principe de l'analyse :

- initialiser les gigues à 0
- Calculer les temps de réponse processeur par processeur et réseau par réseau (systèmes indépendants)
- Mettre à jour les gigues avec les temps de réponse

Itérer les calculs jusqu'au premier point fixe

4.1 Tests Approchés sans garantie

• Chaque système (processeur, réseau) possède sa propre fonction de calcul du pire *temps de réponse*

$$1 \le i \le n \qquad \begin{cases} J_i^{(0)} = 0 \\ R_i^{(k)} = \text{TempsReponse}\left(i, J_i^{(k-1)}\right) \\ J_i^{(k)} = \max_{j \in pred(i)} \left(R_i^{(k)}\right) \end{cases}$$

n : nombre de tâches et de messages
pred(*i*) : prédécesseur immédiat de *i* dans le graphe de communication

4.1 Tests Approchés sans garantie

Temps de calcul :

• Calculs sont rapides
(sous problèmes pseudo-polynomiaux)

• L'ordre de résolution des équations a peu d'influence sur le temps de calcul

Pessimisme :

• Amélioration des bornes (Palencia *et al.*, 1998, 1999, 2003)

• L'analyse holistique permet de valider des systèmes où les processeurs sont chargés à 70% (M. Richard, 2003)

4.2 Tests Approchés avec garantie

- Algorithme approché *A* pour <u>toute</u> instance de problème I (d'optimisation)

$$\frac{\left(A(I) - OPT(I)\right)}{OPT(I)} \leq \varepsilon$$

  - Approximation polynomiale
  - Schéma d'approximation polynomiale (PTAS)
  - Schéma d'approximation polynomiale complet (FTPAS)

Algorithmes où ε est un paramètre d'entrée

4.2. Approximation des temps de réponse

- Approximation polynomiale (avec garantie) des temps de réponse

Aucun résultat n'est connu dans la littérature

4.2. Approximation de la demande processeur

- Ordonnançabilité et approximation
  - Le test répond OK

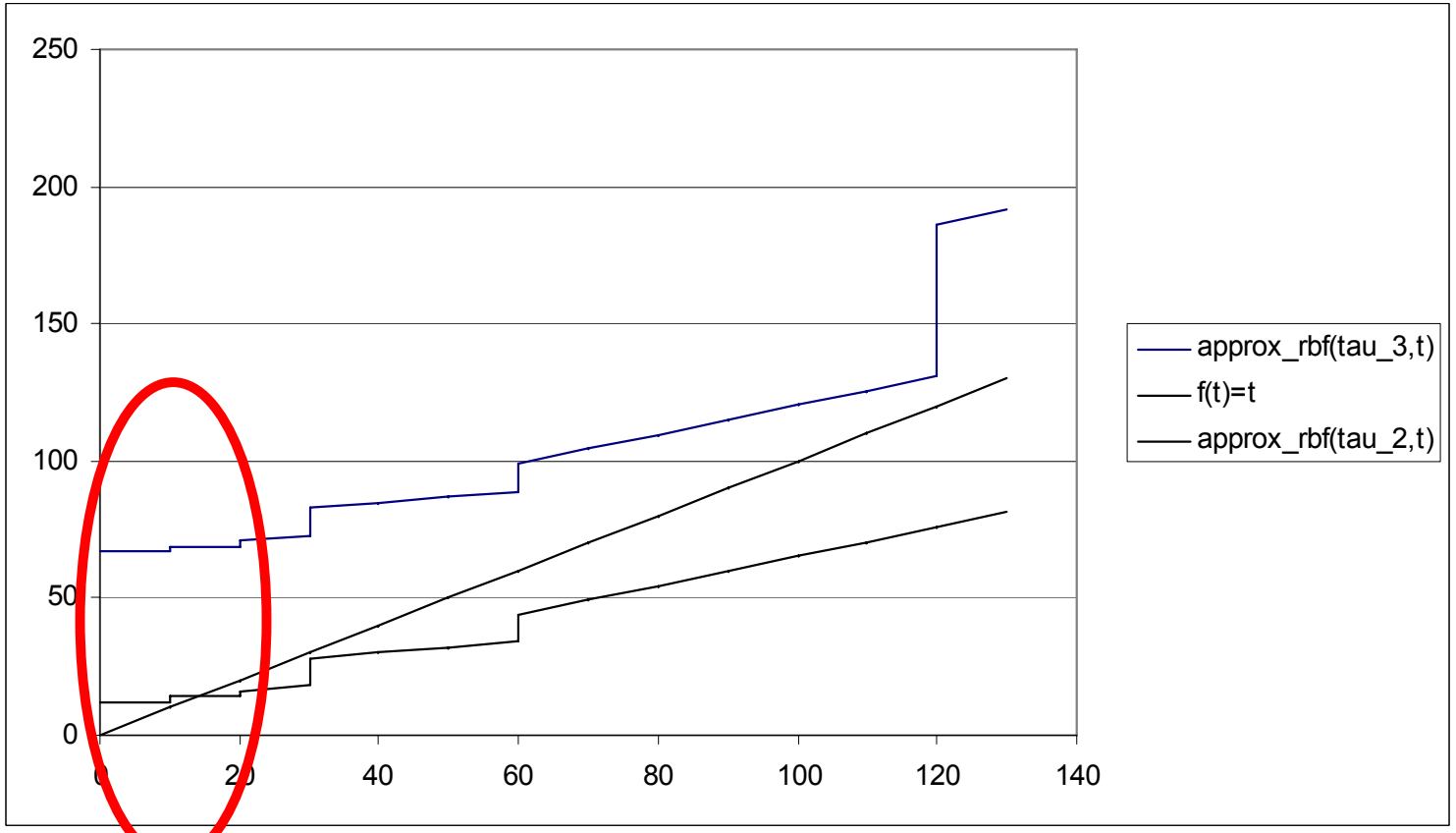
    $\rightarrow$ Syst. Ordonnançable

  - Le test répond $\overline{OK}$

    $\rightarrow$ Syst. non Ordonnançable sur un processeur plus lent (vitesse 1-ε)

4.2. Approximation de la demande processeur

- Approximation de la fonction de travail $W(t)$ (Syst. à priorité fixe – Fisher et Baruah (2004))



On ne considère qu'un nombre limité de paliers, puis
On utilise une borne sup. linéaire

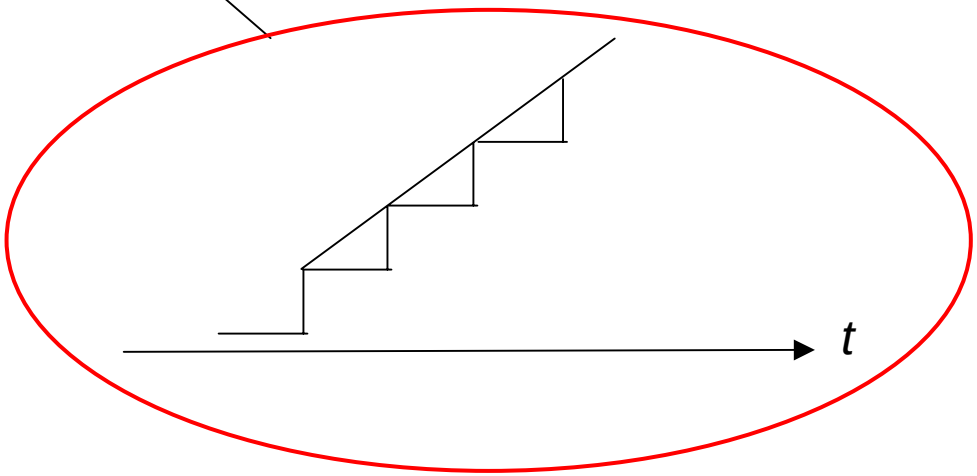
4.2. Approximation de la demande processeur

- FPTAS de Fisher et Baruah (2004), (2005), …

k premiers paliers

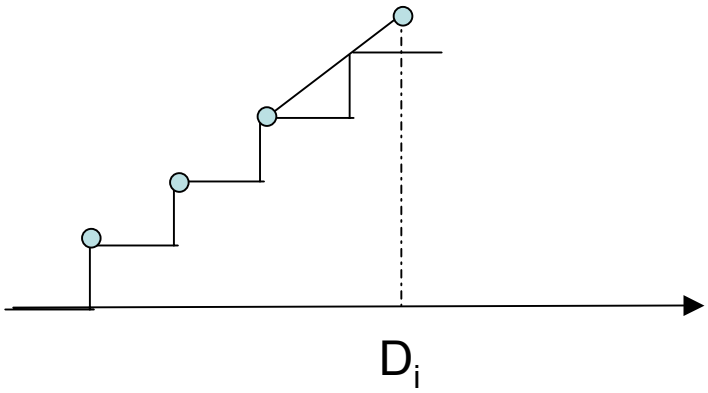
$$\overline{rbf}(0,t) = \begin{cases} rbf(0,t) & t \le (k-1)T_i \\ C_i + t\dfrac{C_i}{T_i} & t > (k-1)T_i \end{cases}$$

Borne supérieure linéaire de la fonction rbf(t)



4.2. Approximation de la demande processeur

- Le test :
  - Utilisation du principe de l'analyse de la demande processeur de Lehoczky, Sha et Ding (1989)
  - Ensemble de test à une taille polynomiale



5.1 Les Simplifications

- Rendre un calcul exact…. s'assurer que le pire scénario peut survenir
  - Ajouter des gigues sur activations
  - Considérer des tâches sporadiques plutôt que périodiques

5.2 Les difficultés

Les deux problèmes à étudier avant de concevoir un test

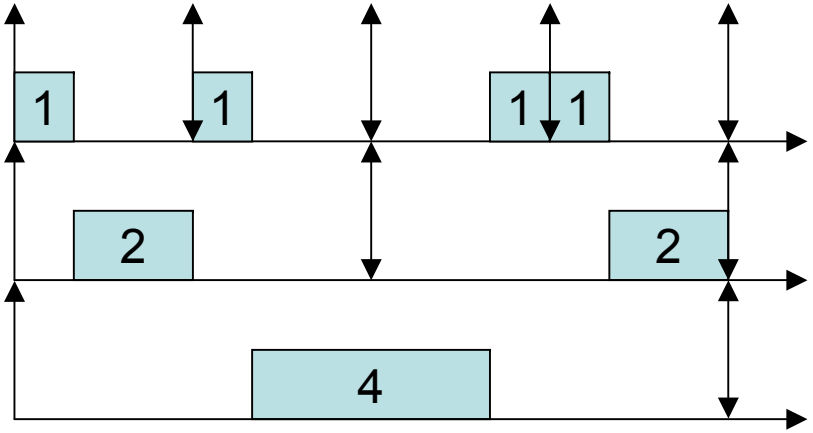
1. Anomalies d'ordonnancement

   « réduire la durée d'une tâche peut rendre le système non ordonnançable »

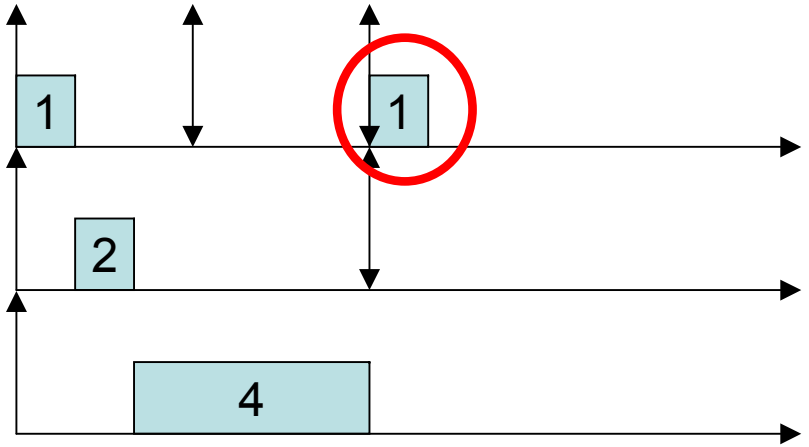
1. Complexité (P, NP, co-NP,…)

5.1. Anomalies

- Ex: ordonnancement non préemptif (FPP)



C_2=2



C_2=1

5.1. Anomalies

- Anomalies en monoprocesseur
  - Ressources partagées
  - Contraintes de précédence en FPP
  - Tâches avec suspension (Entrée/Sortie)

5.1. Anomalies

- Les anomalies rendent :
  - Ordonnancement non robuste
  - Pire scénario est plus difficile à caractériser

5.2. Complexité

- Complexité des problèmes d'ordonnançabilité

Stratégies Dispatching	Caractéristiques des tâches	Complexités
Pmtn, no susp	$D_i=T_i$; $r_i=0$	*P*
Pmtn, no susp	$D_i \neq T_i$; $r_i=0$	ouvert
Pmtn, no susp	$r_i \neq 0$	co-*NP*-Difficile
Pmtn, susp	$D_i=T_i$; $r_i=0$	*NP*-Difficile
No pmtn	—	*NP*-Difficile

5.2. Complexité

- **Un cas troublant** : tâches à départ simultané et à échéance sur requête

- Test en FPP (Lehoczky, Sha et Ding 89)

  – Répondre OK en temps polynomial ⇨ NP

- Test EDF (Baruah, Rosier, Howell 90)

  – Répondre $\overline{OK}$ en temps polynomial ⇨ co-NP

Conclusions

- Analyse du temps de réponse
  - Calcul par approximation successive
  - Nombreuses extensions
  - Pas de tests approchés avec garantie de perf.

- Analyse de la demande processeur
  - Calcul sur un ensemble de test
  - Difficile à étendre à des syst. complexes
  - Tests approchés avec garantie de perf.

Conseils de lecture

- Analyse de la demande processeur
  - EDF : Ripoll, Crespo, Mok « Improvement in feasibility testing for real-time tasks », Journal of Real-Time Systems, 11(1):19-39, 1996
  - FPP : Manabe, Aoyagi « A feasibility decision algorithm for rate monotonic and deadline monotonic scheduling », Journal of Real-Time Systems, 14(2):171-181, 1998

Et encore plus bientôt….

RTNS 2006

Real-Time and Network Systems

30-31 mai 2006

Poitiers

http://www.lisi.ensma.fr/rtns06