Smews : un système d'exploitation dédié au support d'applications Web en environnement contraint

Simon Duquennoy

le 19 juillet 2010

Villeneuve d'Ascq

Rapporteurs :	Didier Donsez	LIG, Université Joseph Fourier – Grenoble 1
	Pierre Sens	LIP6, Université Pierre et Marie Curie – Paris 6
Examinateurs :	Christine Morin	IRISA, Université de Rennes 1
	Laurent Réveillère	LaBRI, Institut Polytechnique de Bordeaux
	Jean-Jacques Vandewalle	CNFPT, École Nationale Supérieure des Sciences de l'Information et des Bibliothèques
Directeur :	Gilles Grimaud	LIFL, Université Lille 1 – Sciences et Technologies

Université Lille1 Sciences et Technologies

cnrs dépasser les frontières

INRIA

Contexte – le Web des objets

L'informatique ambiante

- systèmes informatiques omniprésents
- au besoin de communication grandissant

Avec le Web on sait accéder à l'autre bout du monde mais pas aux objets qui nous entourent...

Les technologies du Web pour les Objets

- simple
- interopérable
- flexible
- déjà standard !

Un véritable verrou technologique

unités : 1,6 million (Google)
CPU : 16 × 2 GHz
données : > 10 Go
stockage : > 10 To
réseau : 10 Gbps

unités : > ×3000
CPU : ×1/4000
données : ×1/1 000 000
stockage : ×1/150 000 000
réseau : ×1/40 000

unités : 5 milliards/an (cartes)
CPU : 4-8 MHz
données : 0,5-10 ko
stockage : 8-64 ko
réseau : 56-256 Kbps

Embarquer des serveurs Web…

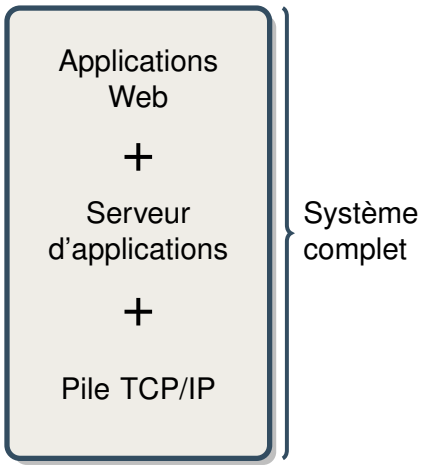
- des technologies exigeantes
- pour les puissants serveurs d'Internet

… dans des matériels contraints

- à faible capacité de calcul
- disposant de peu de mémoire

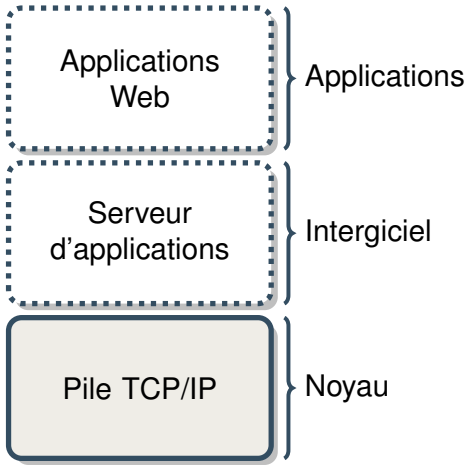
Intéressons nous à la structure même des systèmes d'exploitation utilisés…

Thèse défendue



a) système intégré

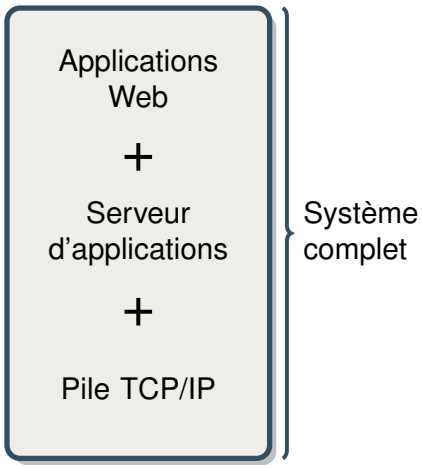
e.g. iPic, webACE, MiniWeb



b) système généraliste

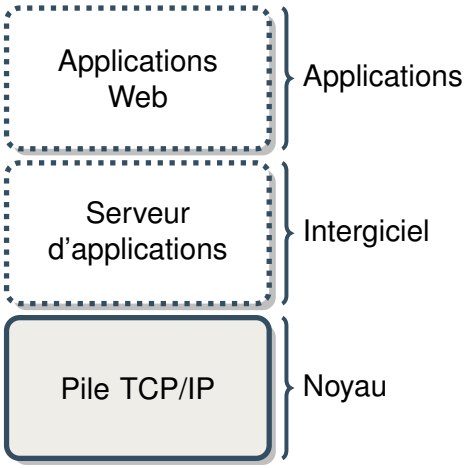
e.g. uIP, MicroNet, μC/TCP-IP

Thèse défendue



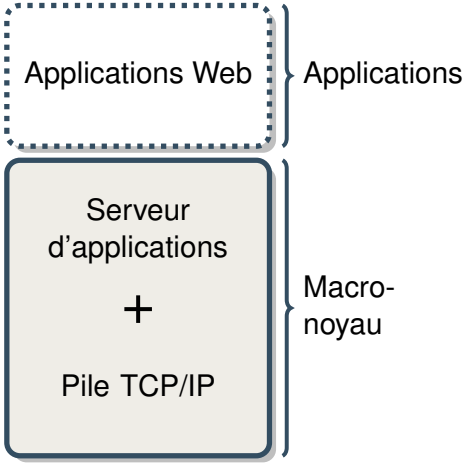
a) système intégré

e.g. iPic, webACE, MiniWeb



b) système généraliste

e.g. uIP, MicroNet, μC/TCP-IP

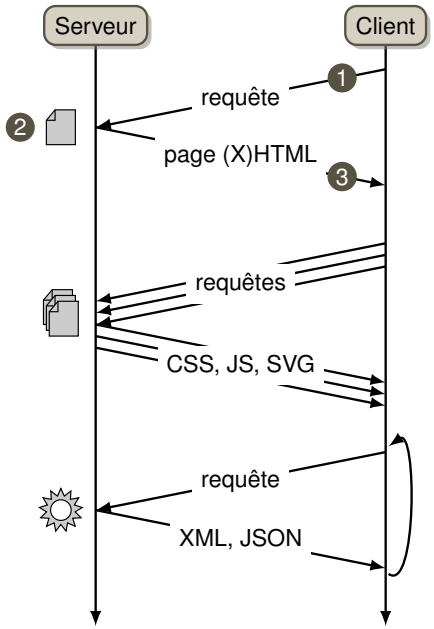


c) système dédié

Vers un système à macro-noyau dédié

- un système dédié à une famille d'applications *(ici, le Web !)*
- une interface applicative de haut niveau
- un noyau spécialisé donc performant

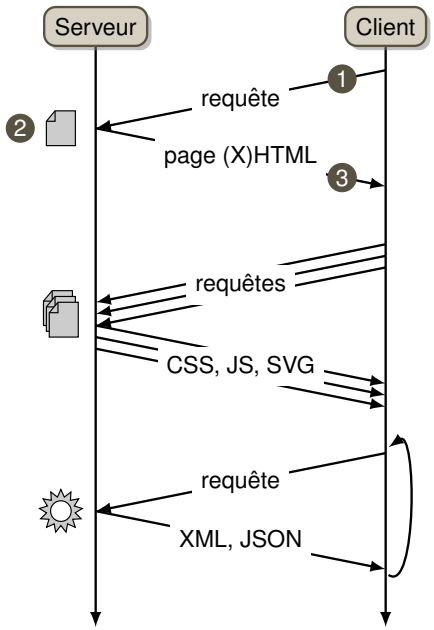
Applications Web et macro-noyau



Fonctionnement des applications Web

1. un client émet une requête (HTTP)
2. le serveur fait appel à une application
   - des fichiers : (X)HTML, CSS, JS...
   - du code applicatif : c, java, perl, ...
3. puis il émet une réponse au client

Applications Web et macro-noyau



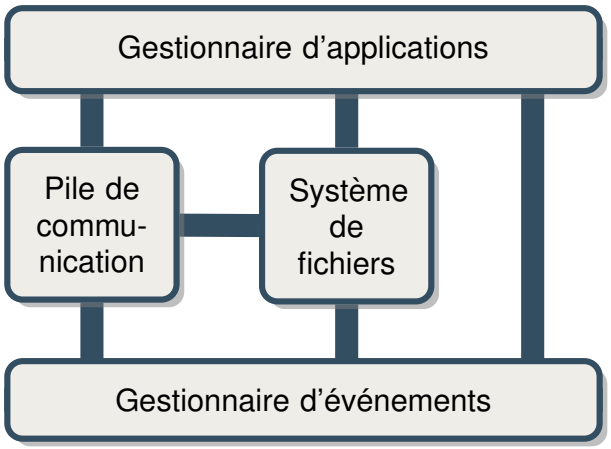
Fonctionnement des applications Web

1. un client émet une requête (HTTP)
2. le serveur fait appel à une application
   - des fichiers : (X)HTML, CSS, JS...
   - du code applicatif : c, java, perl, ...
3. puis il émet une réponse au client

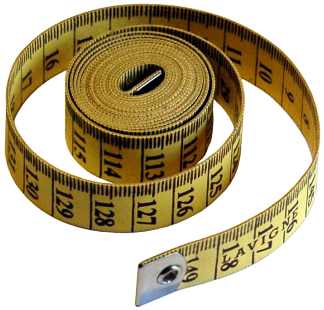
Le macro-noyau, un socle pour applications Web (tierces)

Le rôle du macro-noyau

- gestion des tâches
- gestion des fichiers
- gestion des protocoles
- gestion du matériel



Les métriques clé



Compacité

- taille du code exécutable
- quantité de mémoire requise à l'exécution

Performances

- vitesse d'émission des données
- usage processeur (rapidité, énergie)

La notion de charge mémoire

- intégration de la mémoire sur le temps
- estime la disponibilité
- s'exprime en octet-secondes ($o \cdot s$)

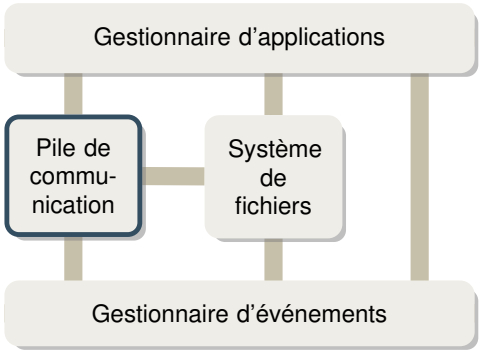
Plan

1. Pile de communication
2. Gestion des tâches applicatives
3. Ordonnancement des requêtes
4. Évaluation d'un système dédié : Smews
5. Conclusion et perspectives

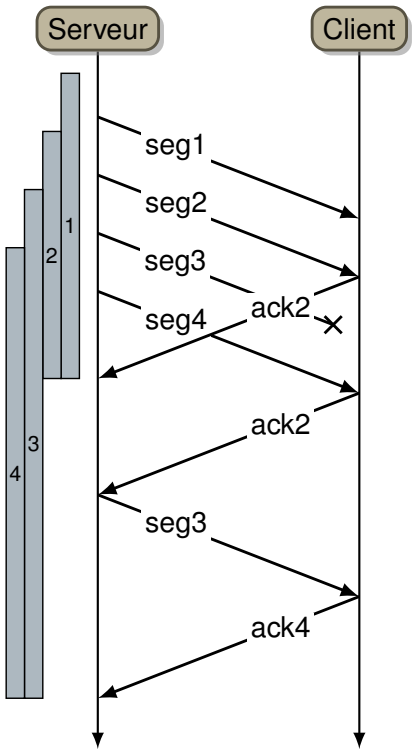
Plan

1. Pile de communication
   - Contexte et état de l'art
   - Proposition
   - Évaluation
2. Gestion des tâches applicatives
3. Ordonnancement des requêtes
4. Évaluation d'un système dédié : Smews
5. Conclusion et perspectives

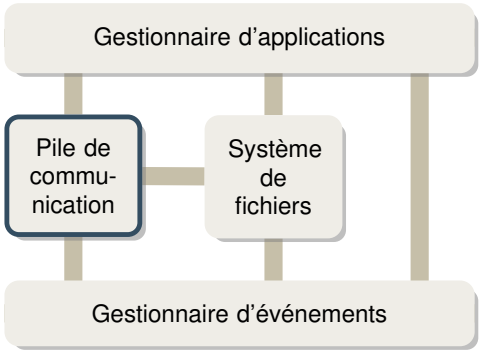
Émission de données TCP



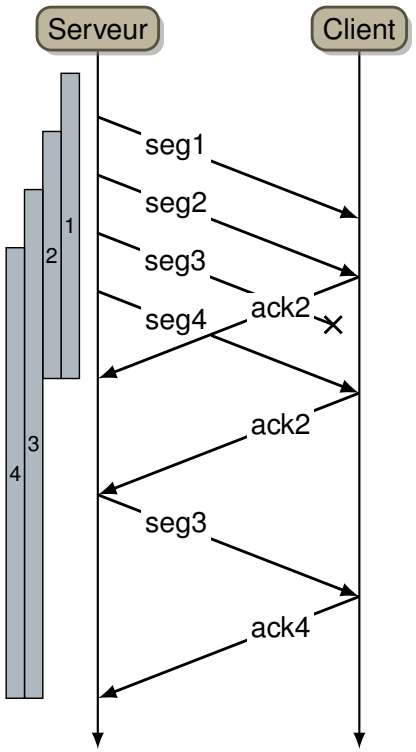
Stratégie classique :
– support naturel de TCP
– rapide
– gourmand en mémoire



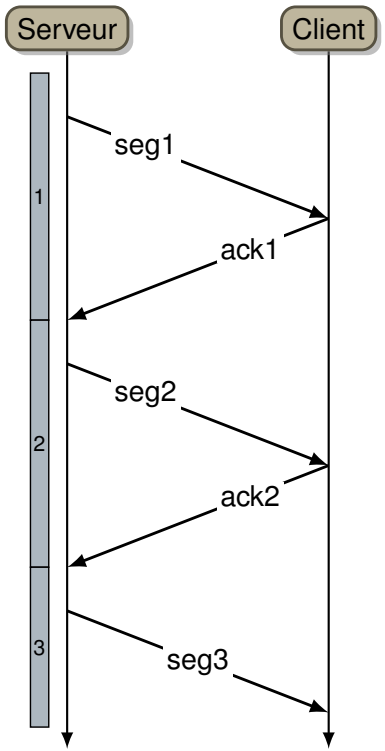
Émission de données TCP



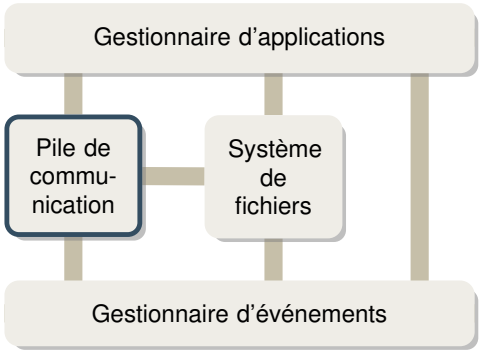
Stratégie classique :
– support naturel de TCP
– rapide
– gourmand en mémoire



Stratégie à un segment :
– uIP [Dunkels, MobiSys'03], TI TCP/IP, . . .
– lent
– adapté à l'embarqué

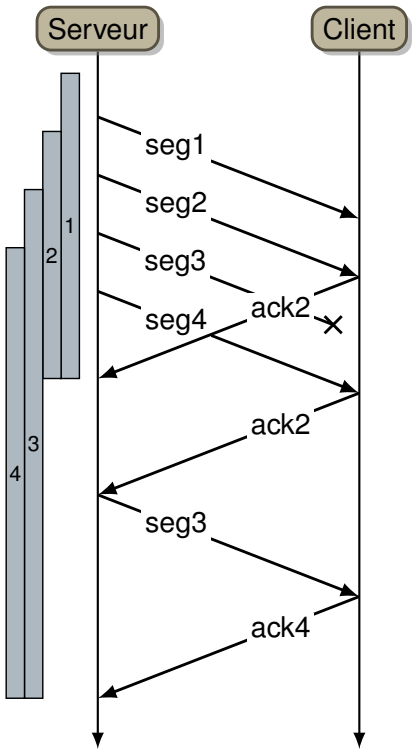


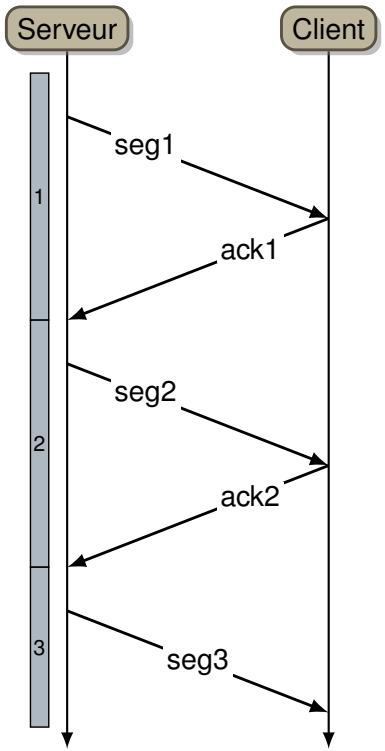
Émission de données TCP

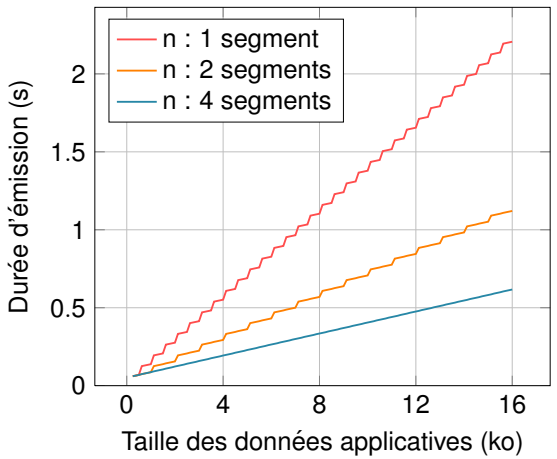


Stratégie classique :
– support naturel de TCP
– rapide
– gourmand en mémoire

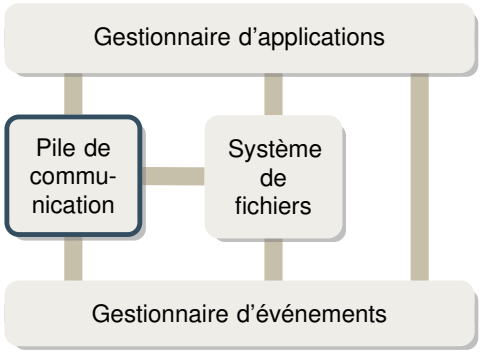
Stratégie à un segment :
– uIP [Dunkels, MobiSys'03], TI TCP/IP, . . .
– lent
– adapté à l'embarqué





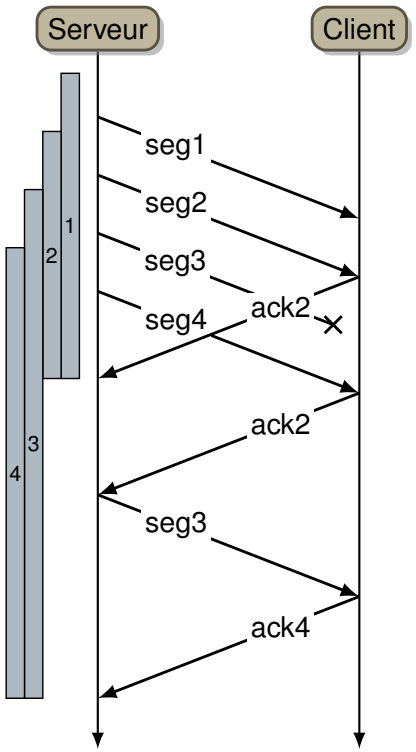


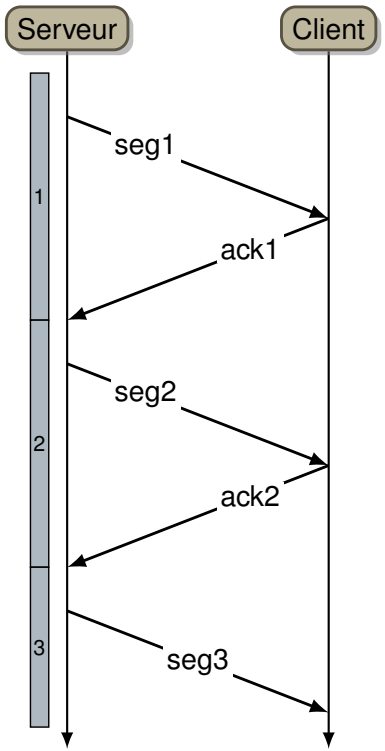
Émission de données TCP

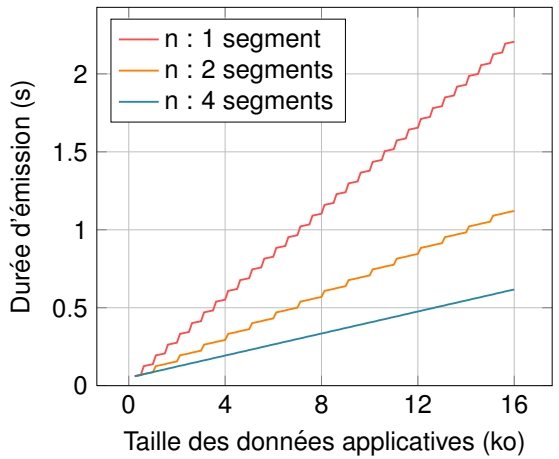


Stratégie classique :
– support naturel de TCP
– rapide
– gourmand en mémoire

Stratégie à un segment :
– uIP [Dunkels, MobiSys'03], TI TCP/IP, . . .
– lent
– adapté à l'embarqué







Et si on s'intéressait à la nature des données applicatives transportées ?

Taxinomie des applications Web

Identifier les propriétés des applications dans le but d'adapter la stratégie d'émission

De quoi sont constituées les applications Web ?

- de contenus statiques (fichiers)
- de contenus dynamiques (code exécuté)

Nature des données applicatives

1. statiques fixées hors-ligne – *fichier*
2. statiques fixées à l'exécution – *log*
3. générées persistantes – *traitement quelconque*
4. générées volatiles – *échantillon*
5. générées idempotentes – *fonction*

L'exemple des données statiques

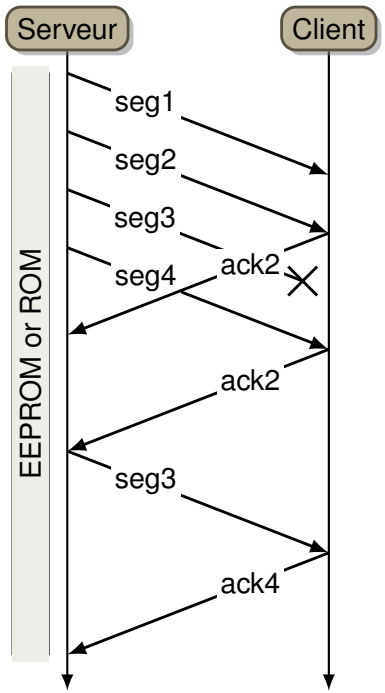
Stratégie d'émission

- émission depuis la mémoire persistante
- facilité de retransmission
- charge mémoire nulle
- grande fenêtre d'émission

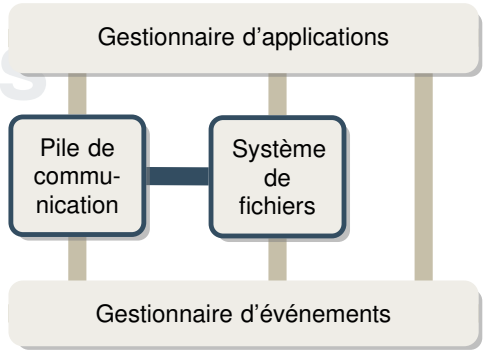
Taille de la fenêtre est bornée par les caractéristiques de la liaison

$$nev_{max} = \left\lceil \frac{2 \times D \times L + MSS + 40 + 40}{MSS + 40} \right\rceil$$

D : débit, L : latence, MSS : taille des segments



L'exemple des données statiques



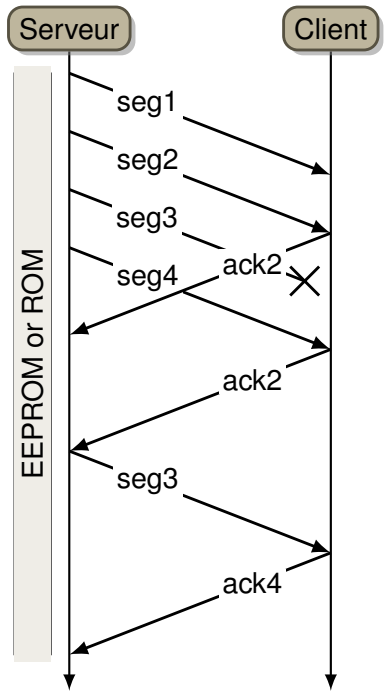
Stratégie d'émission

- émission depuis la mémoire persistante
- facilité de retransmission
- charge mémoire nulle
- grande fenêtre d'émission

Taille de la fenêtre est bornée par les caractéristiques de la liaison

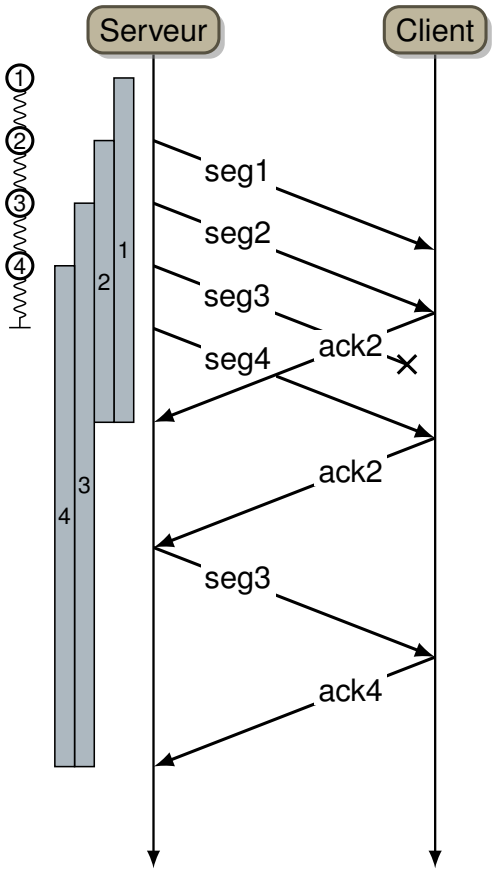
$$nev_{max} = \left\lceil \frac{2 \times D \times L + MSS + 40 + 40}{MSS + 40} \right\rceil$$

D : débit, L : latence, MSS : taille des segments

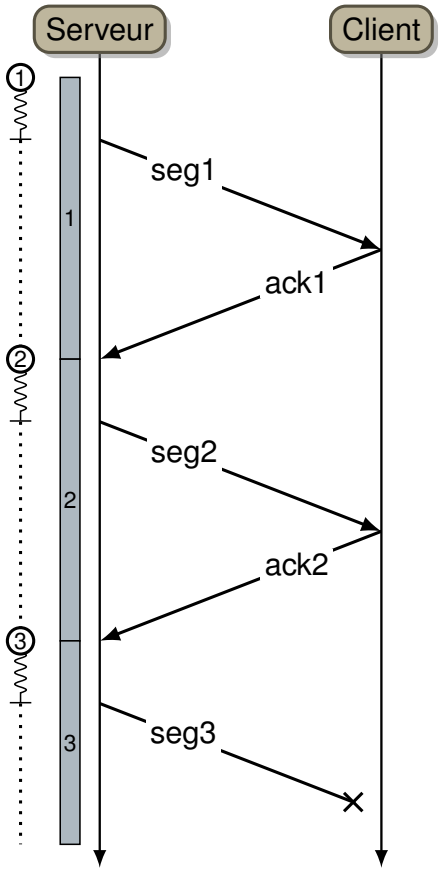


L'exemple des données volatiles

Stratégie classique :
– rapide
– gourmand en mémoire

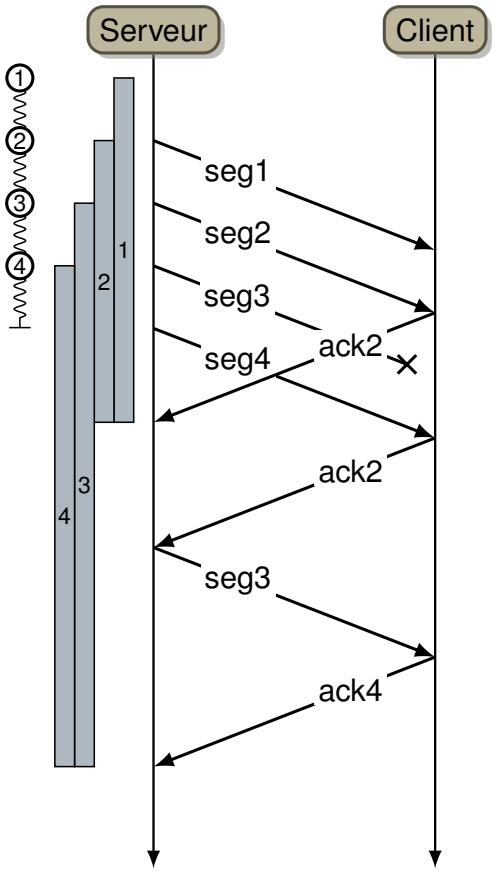


Stratégie à un segment :
– lent
– adapté à l'embarqué

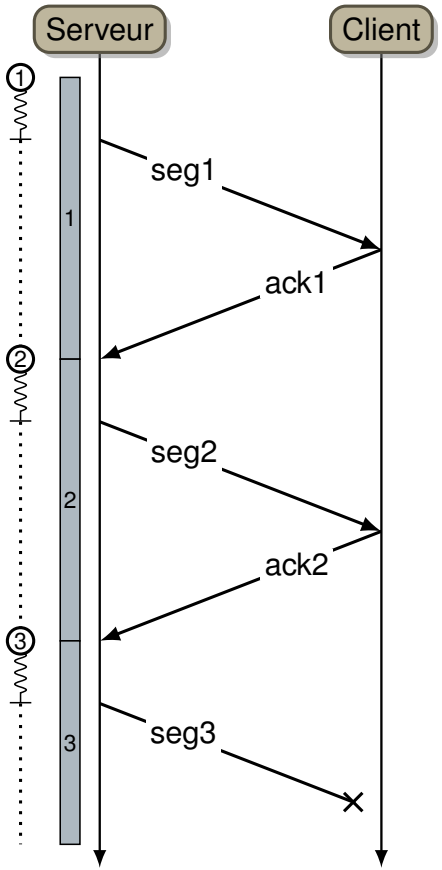


L'exemple des données volatiles

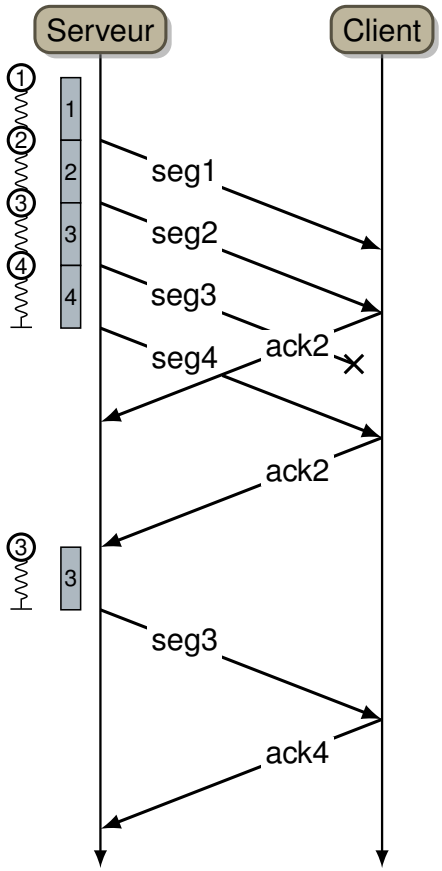
Stratégie classique :
– rapide
– gourmand en mémoire



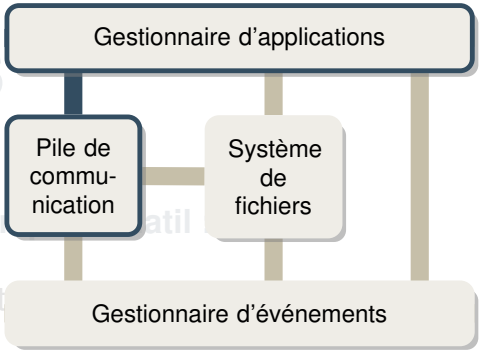
Stratégie à un segment :
– lent
– adapté à l'embarqué



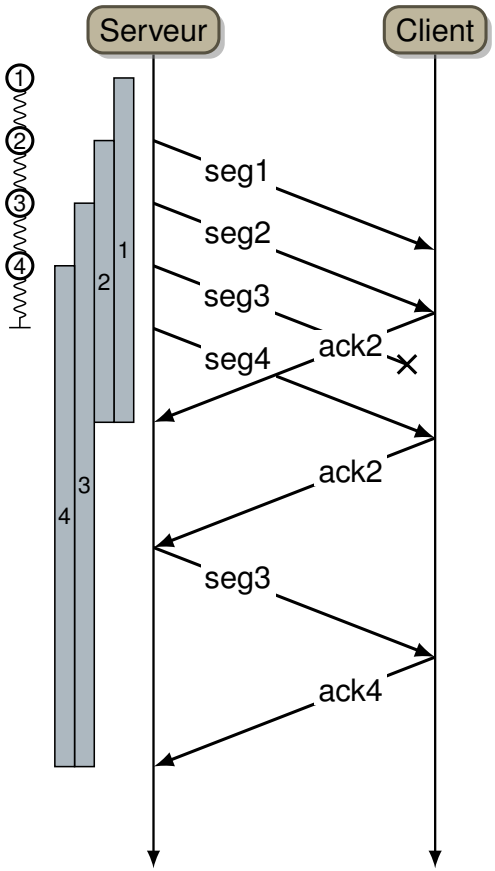
Stratégie du tampon volatil :
– rapide
– 0 à 1 segment en RAM



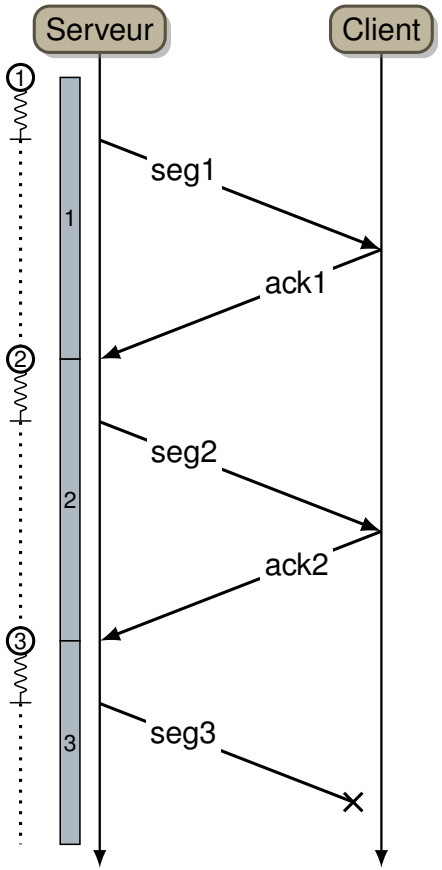
L'exemple des données volatiles



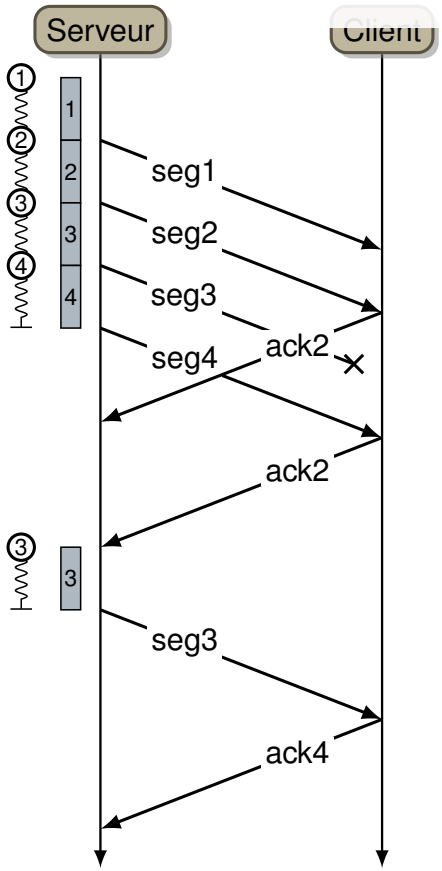
Stratégie classique :
– rapide
– gourmand en mémoire



Stratégie à un segment :
– lent
– adapté à l'embarqué



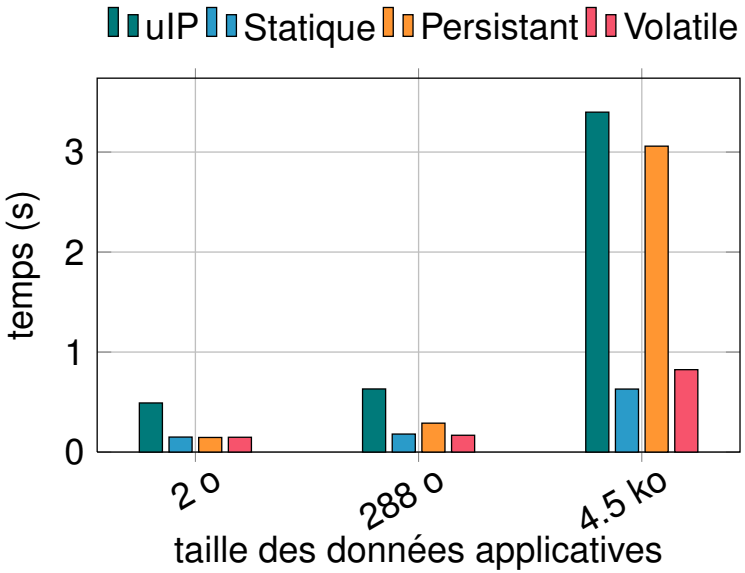
Stratégie du ta[illegible]atil
– rapide
– 0 à 1 segme[illegible]



Performances

Déploiement sur capteur WSN430

- CPU 16 bits MSP430 à 8 MHz
- liaison à 115200 bauds, latence de 50 ms

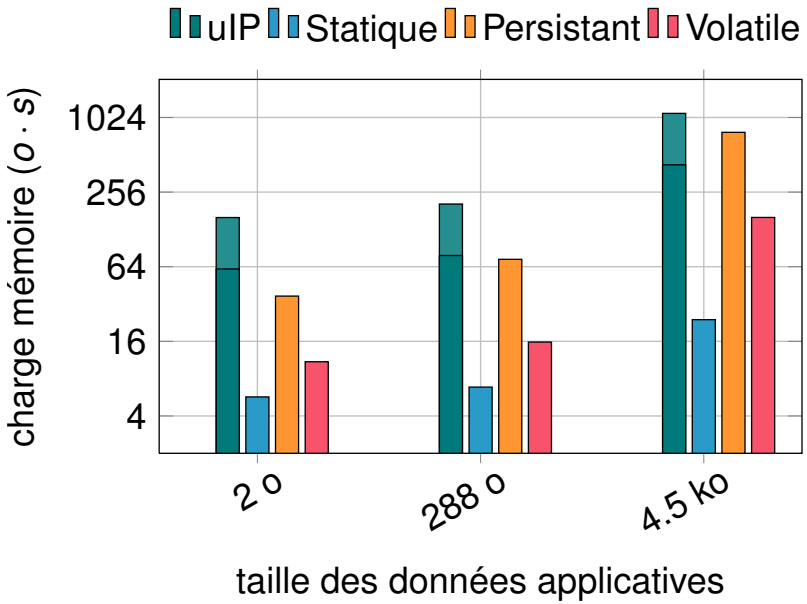


accélération jusqu'à 5,7 fois

Charge mémoire

Toujours sur WSN430

- calculée à partir des performances mesurées…
- et à l'aide de l'accès au code source



réduction dépassant 18 fois

Plan

1 Pile de communication

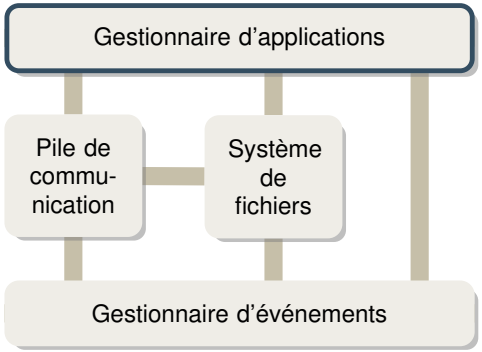
2 Gestion des tâches applicatives
- Contexte et état de l'art
- Proposition
- Évaluation

3 Ordonnancement des requêtes

4 Évaluation d'un système dédié : Smews

5 Conclusion et perspectives

Le gestionnaire d'applications



Gère des tâches applicatives

- constituant les applications Web
- provenant de tiers
- concurrentes

En charge de…

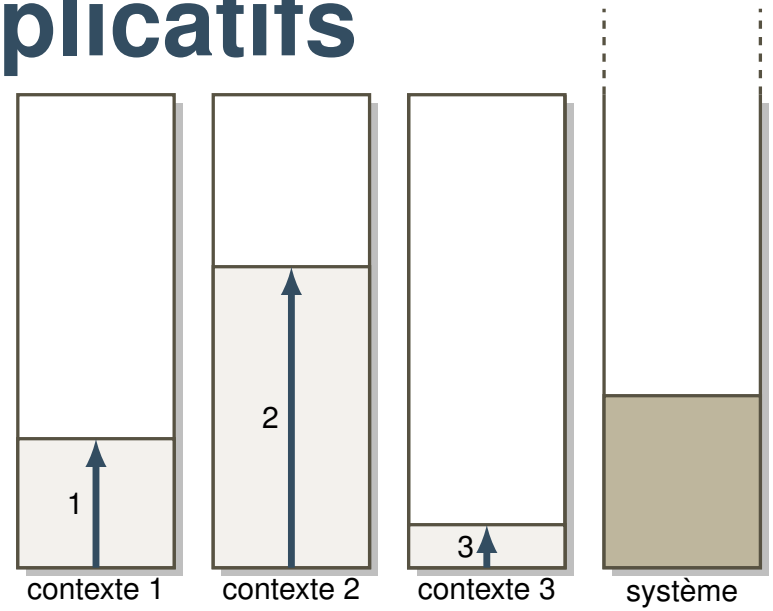
- lancer
- bloquer
- débloquer
- ré-invoquer

Comment exécuter les tâches en maîtrisant la consommation de mémoire ?

Gestion des contextes applicatifs

Coroutines

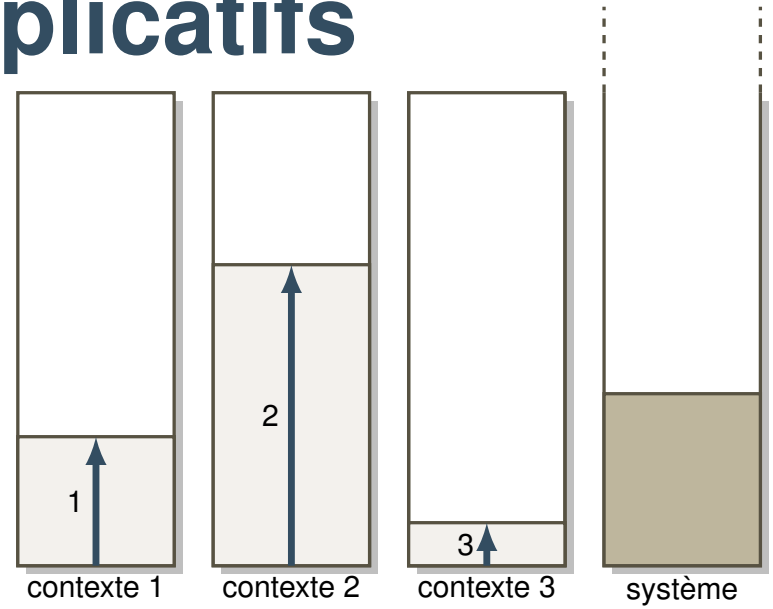
- une pile par tâche
- taille de pile statique
  - *e.g.* 128 octets dans MantisOS [Bhatti *et al.*, MONET, 2005]



Gestion des contextes applicatifs

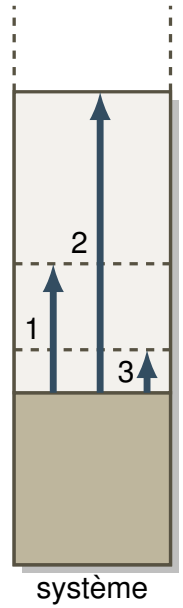
Coroutines

- une pile par tâche
- taille de pile statique
  - *e.g.* 128 octets dans MantisOS [Bhatti *et al.*, MONET, 2005]



Protothreads [Dunkels *et al.*, Sensys'06]

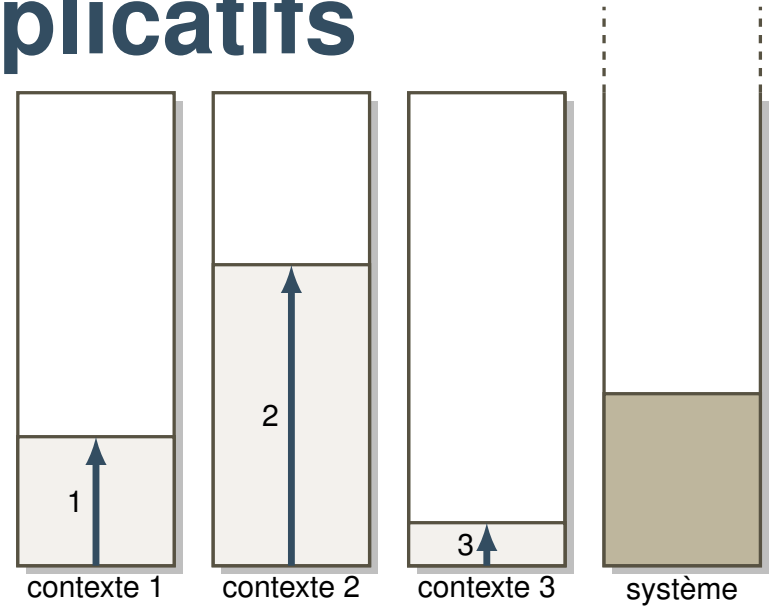
- des événements outillés
- de fortes contraintes sur les applications
  - absence de contexte persistant (locales)
  - imbrications explicites
  - absence de bloc `switch`



Gestion des contextes applicatifs

Coroutines

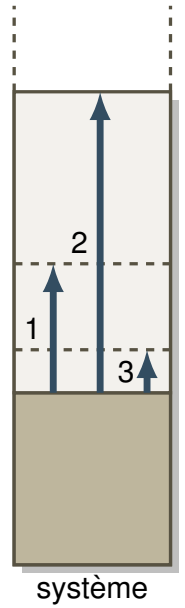
- une pile par tâche
- taille de pile statique
  - *e.g.* 128 octets dans MantisOS [Bhatti *et al.*, MONET, 2005]



Protothreads [Dunkels *et al.*, Sensys'06]

- des événements outillés
- de fortes contraintes sur les applications
  - absence de contexte persistant (locales)
  - imbrications explicites
  - absence de bloc `switch`

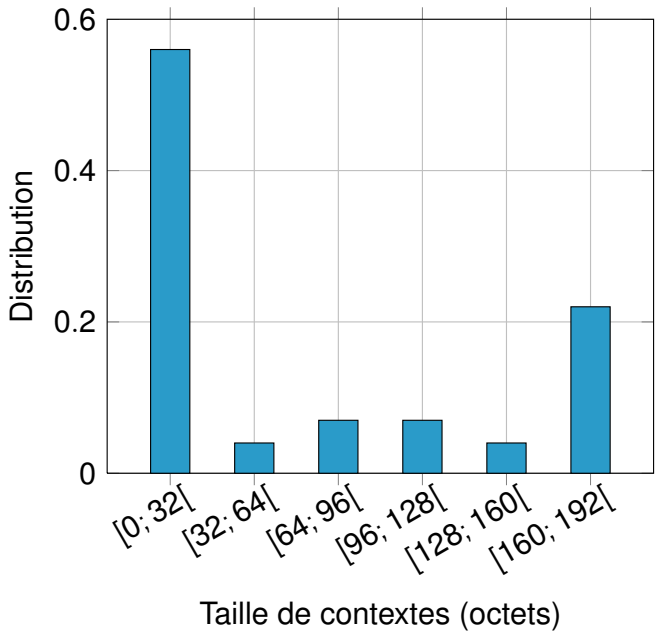
Les protothreads ne sont pas adaptés, quel est le coût réel des coroutines ?



Caractérisation des tailles de contextes

Mesurer la taille des contextes applicatifs

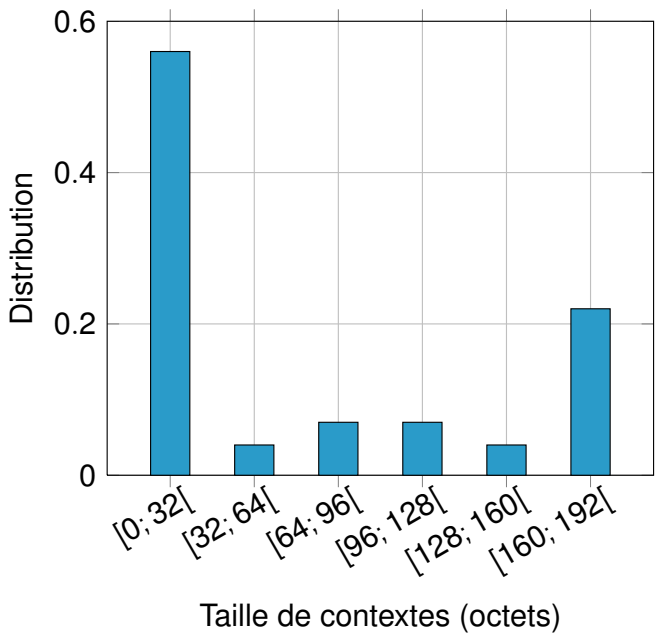
- outillage du système Contiki
- exécution d'un ensemble d'applications
- enregistrement du pointeur de pile (blocages, déblocages)



Caractérisation des tailles de contextes

Mesurer la taille des contextes applicatifs

- outillage du système Contiki
- exécution d'un ensemble d'applications
- enregistrement du pointeur de pile (blocages, déblocages)



Conclusions

- taille très variable
- médiane très faible : 26 octets

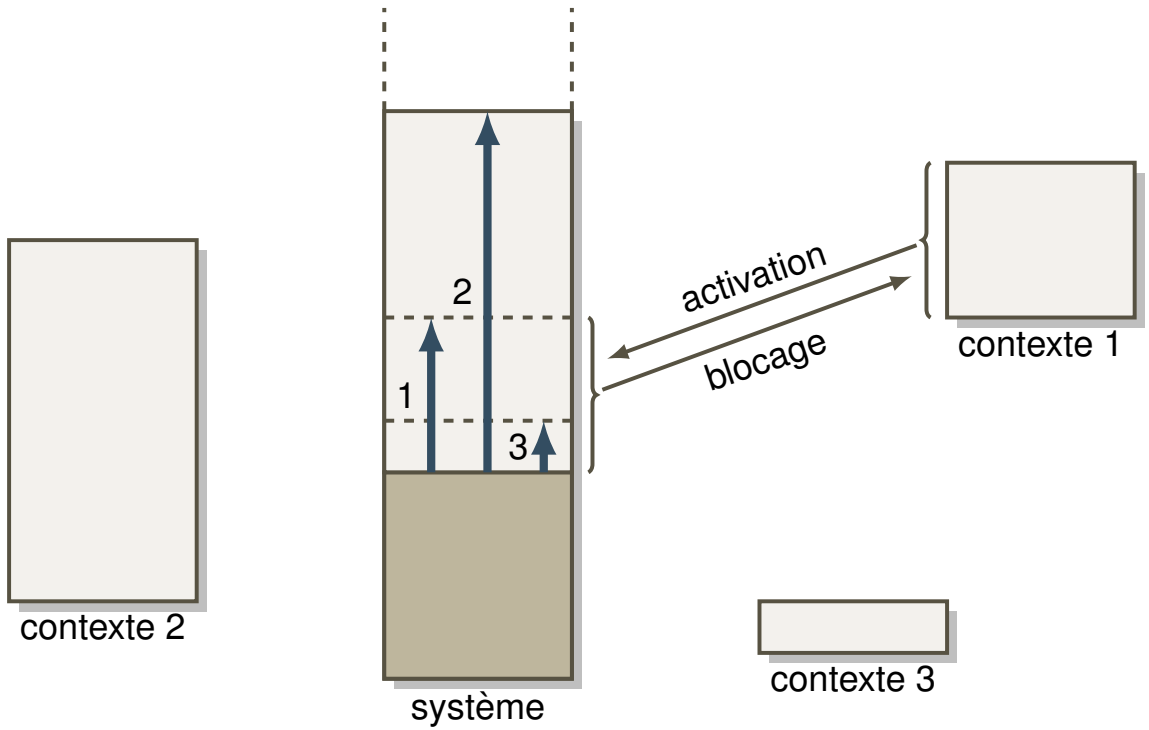
Il faut s'adapter à la taille des contextes applicatifs

Une solution souple et transparente

Proposition de coroutines à pile partagée

- sauvegarde de contexte à chaque blocage
- restauration de contexte à chaque déblocage
- contextes des appels non bloquants non sauvegardés

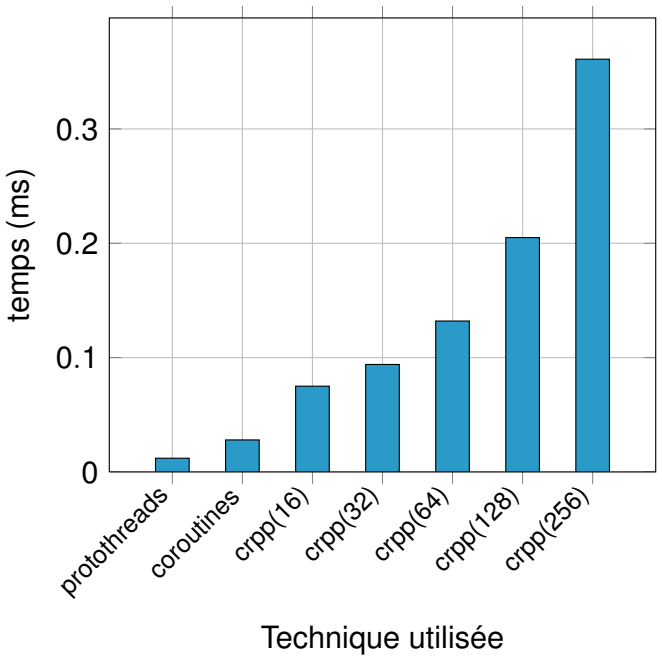
La ré-invocation est naturellement permise



Bénéfices et compromis

Bénéfices des coroutines à pile partagée

- invisible pour l'application
- consommation mémoire faible
- ré-invocations supportées aisément



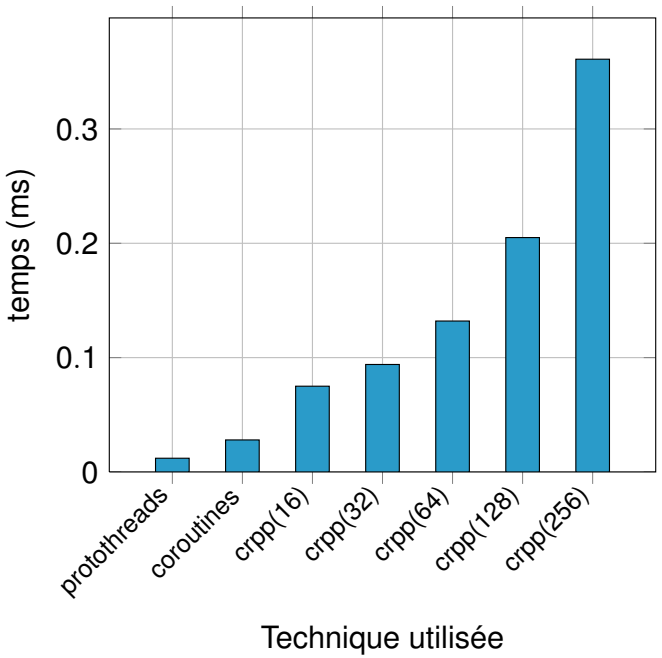
Caractérisation des surcoûts

- 0,08 à 0,36 ms
  (sur MSP430 à 8 MHz)
- à rapporter aux 2,5 ms par segment dans uIP

Bénéfices et compromis

Bénéfices des coroutines à pile partagée

- invisible pour l'application
- consommation mémoire faible
- ré-invocations supportées aisément



Caractérisation des surcoûts

- 0,08 à 0,36 ms
  (sur MSP430 à 8 MHz)
- à rapporter aux 2,5 ms par segment dans uIP

Comment ordonnancer efficacement les tâches ?

Plan

1 Pile de communication

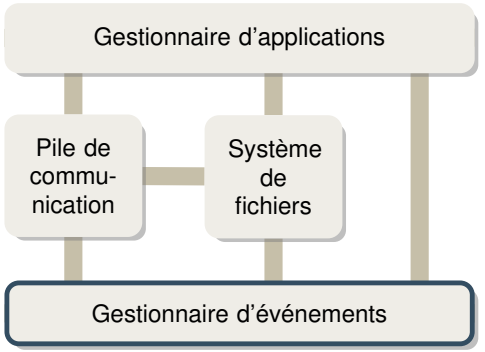
2 Gestion des tâches applicatives

3 Ordonnancement des requêtes
- Contexte et état de l'art
- Proposition
- Évaluation

4 Évaluation d'un système dédié : Smews

5 Conclusion et perspectives

Ordonnancement



Ordonnancement dans les serveurs Web

- gérer l'ordre de service des requêtes
- pour des performances, de l'équité…

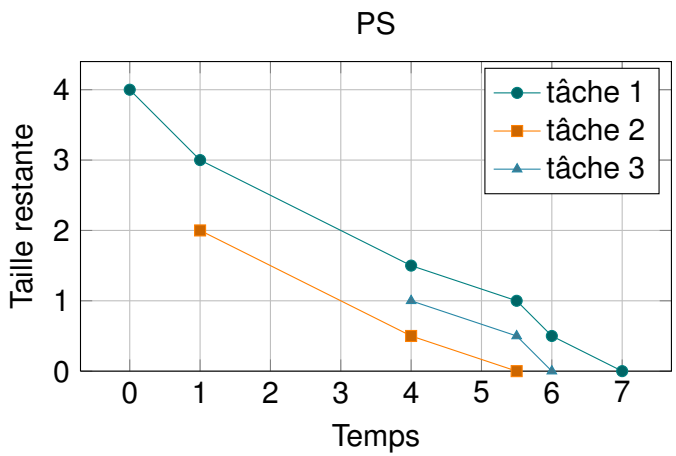
La politique PS

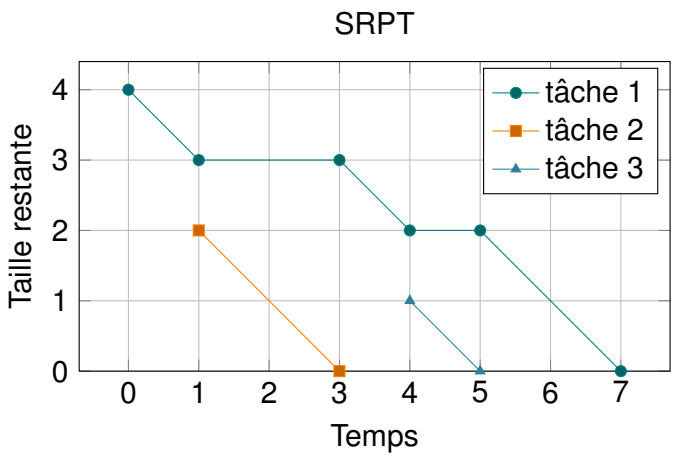
- le tourniquet
- utilisée partout

La politique SRPT

- priorité à la requête dont la taille restante est la plus faible
- prouvée optimale en performances [Scharge, Op. Research, 1968]

Analyse des solutions existantes





Alors, pourquoi SRPT n'est-il pas utilisé ?

- PS partage efficacement les ressources
- PS évite la famine de certaines requêtes
- les pages Web s'affichent graduellement…

Il est donc pertinent de prendre en compte l'équité

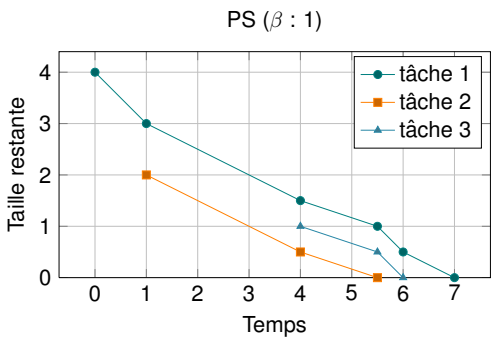
La politique β-SRPT

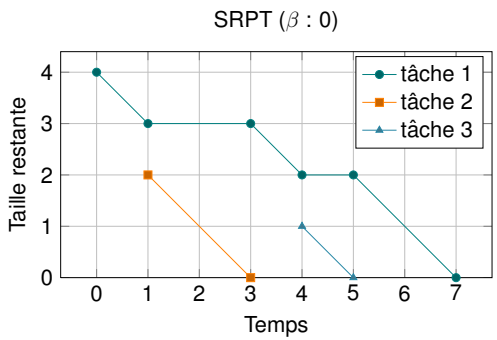
Idée générale

- partage des ressources pondéré
- favorise les tâches courtes, comme SRPT
- paramétré par β: SRPT si $\beta = 0$, PS si $\beta = 1$

$$w_j = \frac{\beta^j}{\sum_{i=0}^{N-1} \beta^i} = \begin{cases} \frac{1}{N} & \beta = 1 \\ \beta^j \times \frac{1-\beta}{1-\beta^N} & \beta \neq 1 \end{cases}$$

Tâches j triées par temps restant croissant, système avec N tâches





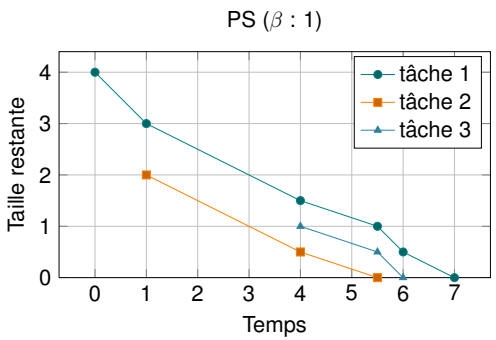
La politique β-SRPT

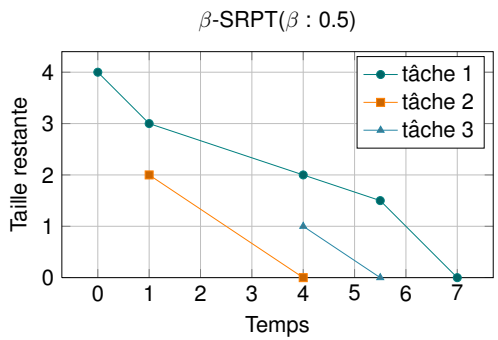
Idée générale

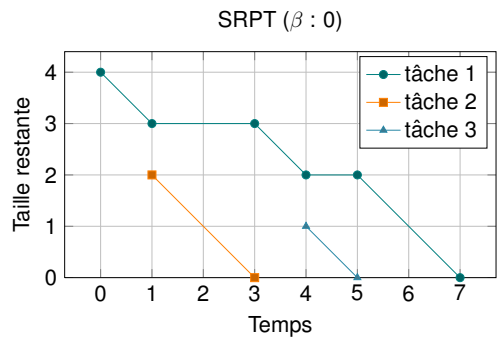
- partage des ressources pondéré
- favorise les tâches courtes, comme SRPT
- paramétré par β: SRPT si $\beta = 0$, PS si $\beta = 1$

$$w_j = \frac{\beta^j}{\sum_{i=0}^{N-1} \beta^i} = \begin{cases} \frac{1}{N} & \beta = 1 \\ \beta^j \times \frac{1-\beta}{1-\beta^N} & \beta \neq 1 \end{cases}$$

Tâches j triées par temps restant croissant, système avec N tâches







Extension au cas pondéré

Objectifs

- étendre β-SRPT au cas pondéré
- poids des tâches : taille de contexte
- minimiser la charge mémoire globale :

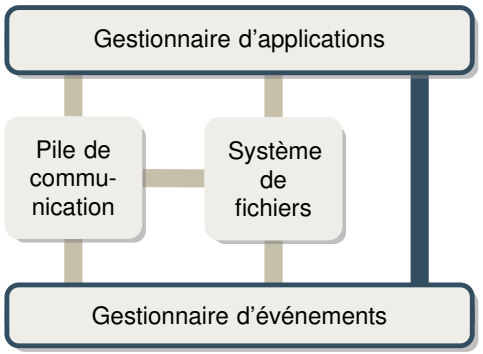
$$B = \int_0^{\infty} \sum_{j \in A(t)} m_j, dt$$

$A(t)$: tâches dans le système au temps t ; m_j : mémoire consommée par la tâche j

La politique β-WSRPT

- intermédiaire entre WSRPT (SRPT pondéré) et PS
- tâches triées par $\frac{R_j}{m_j}$ (R_j : taille restante)
- attribution des poids équivalente à β-SRPT

Extension au cas pondéré



Objectifs

- étendre β-SRPT au cas pondéré
- poids des tâches : taille de contexte
- minimiser la charge mémoire globale :

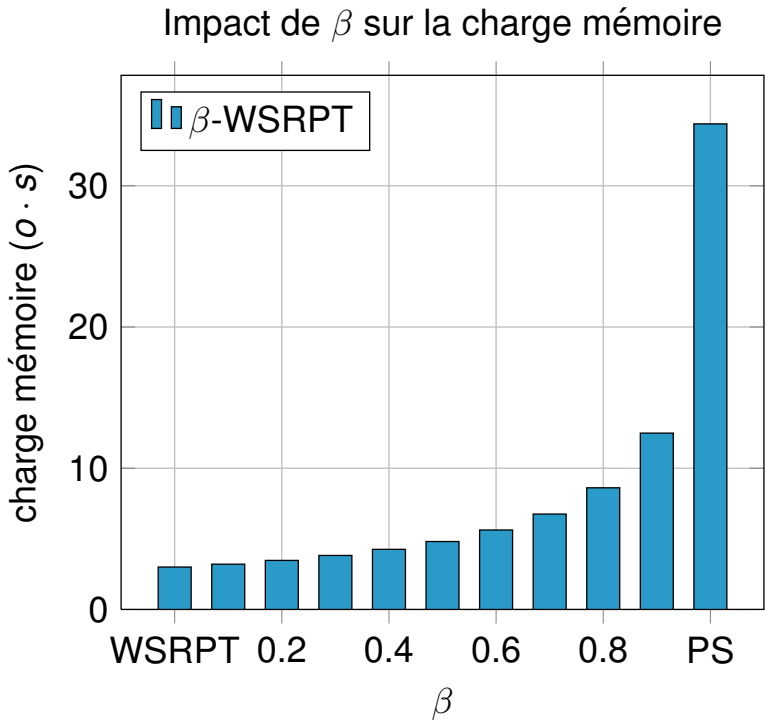
$$B = \int_0^{\infty} \sum_{j \in A(t)} m_j, dt$$

$A(t)$: tâches dans le système au temps t ; m_j : mémoire consommée par la tâche j

La politique β-WSRPT

- intermédiaire entre WSRPT (SRPT pondéré) et PS
- tâches triées par $\frac{R_j}{m_j}$ (R_j : taille restante)
- attribution des poids équivalente à β-SRPT

Résultats de simulations

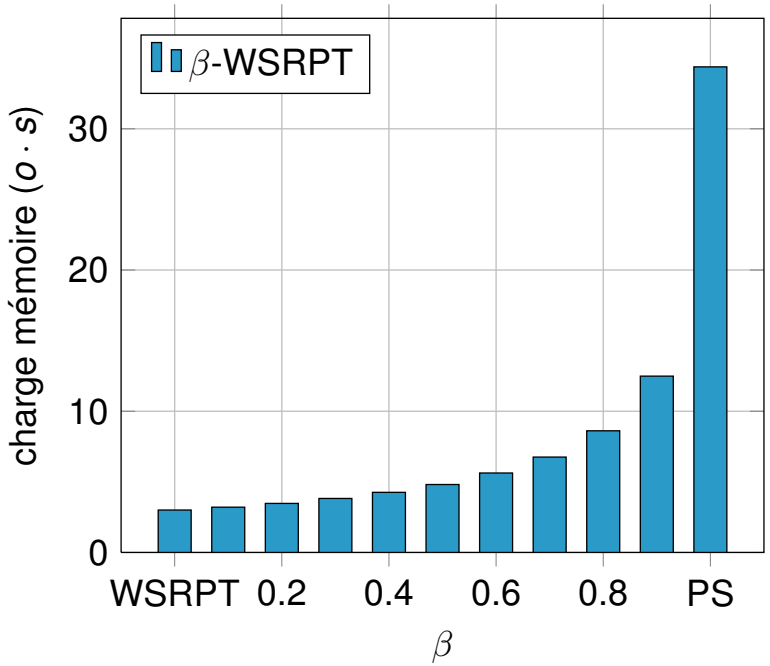


Observations

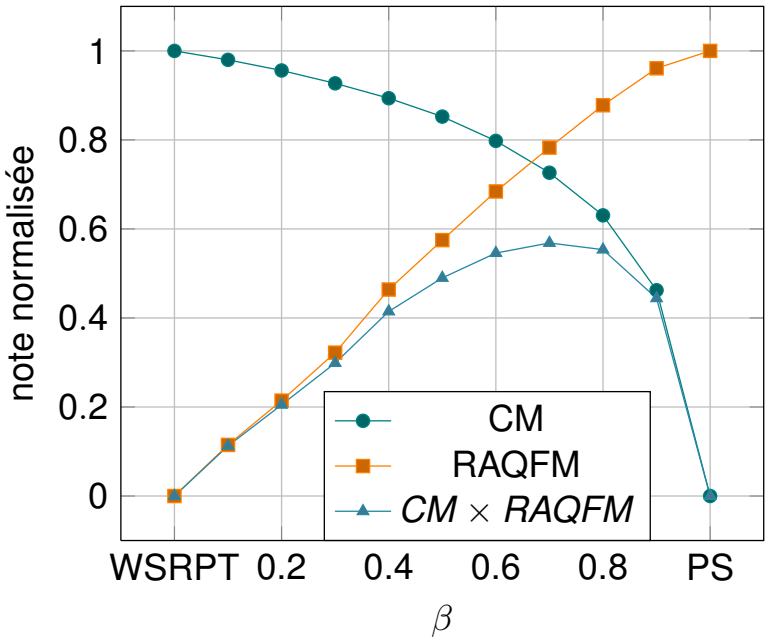
- WSRPT est jusqu'à 10 fois plus léger que PS

Résultats de simulations

Impact de β sur la charge mémoire



Compromis entre charge mémoire et équité



On mesure ici l'équité avec la métrique RAQFM basée sur la notion de discrimination individuelle [Raz *et al.*, SIGMETRICS'04]

Observations

- WSRPT est jusqu'à 10 fois plus léger que PS
- compromis intéressants possibles

Plan

1 Pile de communication

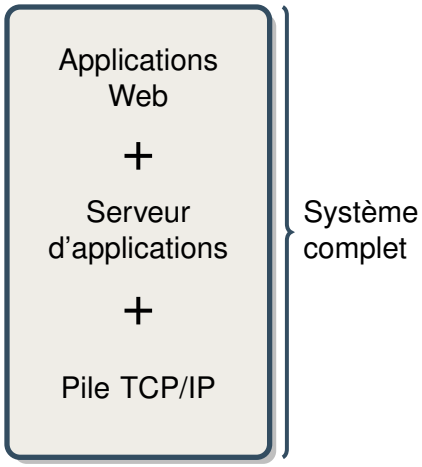
2 Gestion des tâches applicatives

3 Ordonnancement des requêtes

4 Évaluation d'un système dédié : Smews

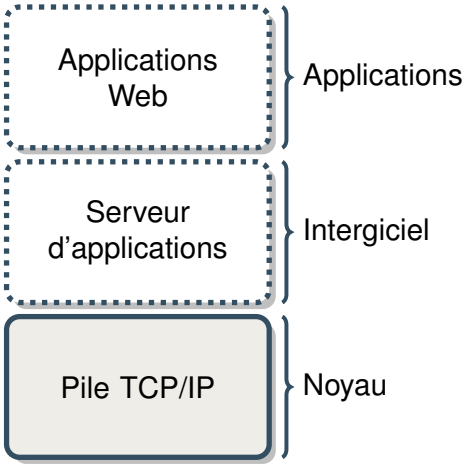
5 Conclusion et perspectives

Smews : un macro-noyau pour un micro-serveur



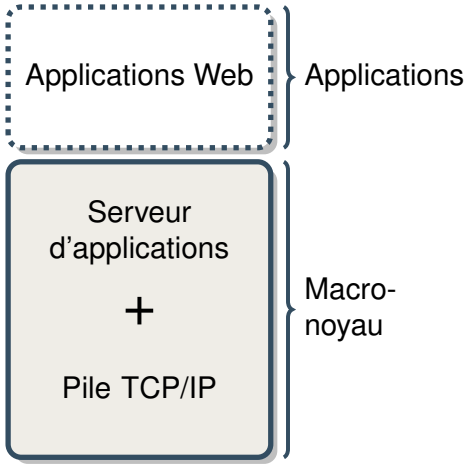
a) système intégré

MiniWeb



b) système généraliste

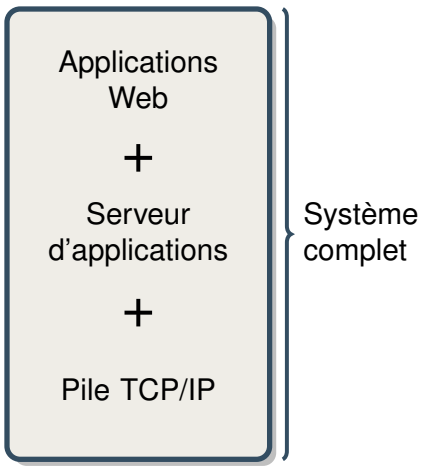
uIP



c) système dédié

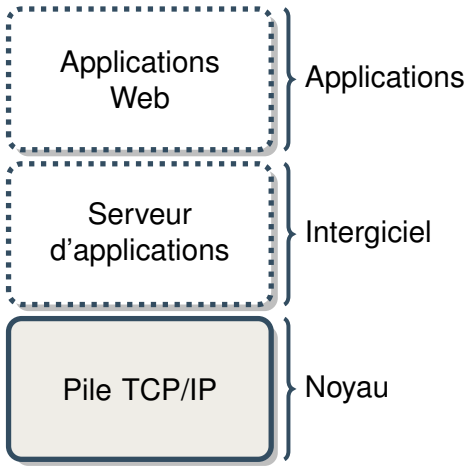
Thèse défendue

Smews : un macro-noyau pour un micro-serveur



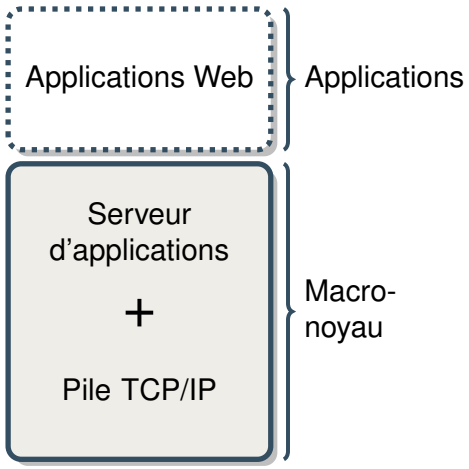
a) système intégré

MiniWeb



b) système généraliste

uIP

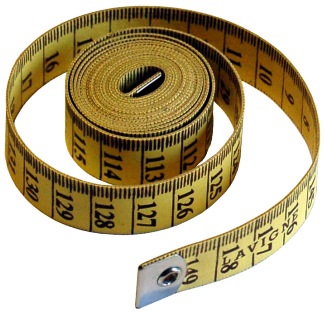


c) système dédié

Thèse défendue

- preuve de concept de l'approche présentée
- logiciel libre (licence CeCILL)
- porté sur des cibles 8, 16 et 32 bits

Comparaison des empreintes mémoire



	mémoire volatile (octets)			mémoire persistante (octets)		
Seveur	pile	données	total vol.	code	données	total pers.
Microprocesseur MSP430 16 bits						
MiniWeb	36	62	**98**	3 k	710	**3,7 k**
Smews	100	174	**274**	7,8 k	240	**8,1 k**
uIP	156	834	**990**	10,8 k	1,2 k	**11,2 k**
Microprocesseur AVR 8 bits						
MiniWeb	52	52	**104**	3,7 k	696	**4,3 k**
Smews	118	172	**274**	9,7 k	240	**9,9 k**
uIP	184	803	**987**	12,4 k	908 k	**13,3 k**

Taille du code embarqué

- MiniWeb : borne inférieure mais sans support applicatif
- noyau de uIP sans routage ni fragmentation : 8.9 ko
- noyau de Smews avec toutes optimisations : 7.8 ko

+ noyau configurable (noyau dur : 61 %)
+ expansion du noyau évitée

Capacité de passage à l'échelle

Méthodologie

- réutilisation des métriques de [Bozdag *et al.*, JWE, 2009]

Exécution de 10 x 252 expérimentations...

- nombres de clients concurrents : [1;256]
- intervalle de publication : [1;50]
- modèle d'interaction : *polling*, alerte, flux
- intervalle de *polling* : [1;50]

Résultats

- jusqu'à 256 clients dans 10 ko RAM
- publications toutes les 15 secondes : 92 % de cohérence

Plan

1 Pile de communication

2 Gestion des tâches applicatives

3 Ordonnancement des requêtes

4 Évaluation d'un système dédié : Smews

5 Conclusion et perspectives

Conclusion

Université Lille1 — CNRS — INRIA

N° d'ordre : 40306

THÈSE

présentée le 19 juillet 2010
pour l'obtention du titre de
Docteur de l'Université Lille 1 – Sciences et Technologies
(en spécialité informatique)

par
Simon DUQUENNOY

Smews : un système d'exploitation dédié
au support d'applications Web
en environnement contraint

Composition du jury :

Rapporteurs : Didier DONSEZ — LIG, Université Joseph Fourier - Grenoble 1
Pierre SENS — LIP6, Université Pierre et Marie Curie - Paris 6
Examinateurs : Christine MORIN — IRISA, Université de Rennes 1
Laurent RÉVEILLÈRE — LaBRI, Institut Polytechnique de Bordeaux
Jean-Jacques VANDEWALLE — CNIPT, École Nationale Supérieure des Sciences de l'Information et des Bibliothèques
Directeur : Gilles GRIMAUD — LIFL, Université Lille 1 - Sciences et Technologies

Thèse – un système d'exploitation dédié

- fournir une interface de haut niveau
- supportée par un macro-noyau intégré

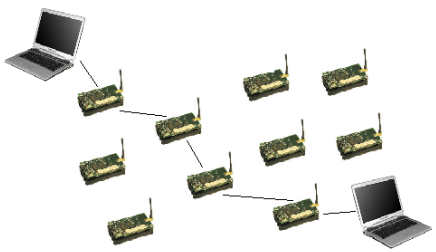
Contributions principales

ICESS'09	Conception transversale HTTP/TCP/IP
EMSOFT'09	Adaptation à la nature des applications ✓
WSE'09	Notification d'événements
MASCOTS'10	Ordonnancement des requêtes ✓
Non publié	Gestion des contextes applicatifs et ré-invocation ✓

Enseignements

- gains les plus importants permis par des refontes radicales
- expansion du noyau évitée (portabilité, maintenabilité)
- chaque famille d'applications nécessite un macro-noyau dédié

Perspectives



Réseaux *ad hoc* sans fil

- pertes et variations de latence
- efficacité des stratégies présentées ?

Conception des applications Web

- exploiter la taxinomie pour outiller la conception
- affiner la granularité pour la nature des données

Application aux stations serveurs

- exploiter les gains en charge mémoire
- adaptation aux multi-cœurs

Autres familles d'application

- *e.g.* systèmes à machine virtuelle, base de données
- bénéfices sur la mémoire virtuelle ou le système de fichiers

Smews : un système d'exploitation dédié au support d'applications Web en environnement contraint

Simon Duquennoy

le 19 juillet 2010

Villeneuve d'Ascq

Rapporteurs :	Didier Donsez	LIG, Université Joseph Fourier – Grenoble 1
	Pierre Sens	LIP6, Université Pierre et Marie Curie – Paris 6
Examinateurs :	Christine Morin	IRISA, Université de Rennes 1
	Laurent Réveillère	LaBRI, Institut Polytechnique de Bordeaux
	Jean-Jacques Vandewalle	CNFPT, École Nationale Supérieure des Sciences de l'Information et des Bibliothèques
Directeur :	Gilles Grimaud	LIFL, Université Lille 1 – Sciences et Technologies

Université Lille1 Sciences et Technologies

cnrs dépasser les frontières

INRIA