Services de communication répartis dans le système Masix

Franck Mével et Julien Simon
Laboratoire MASI, Institut Blaise Pascal
Université Paris VI
4 place Jussieu
75252 PARIS CEDEX 05, FRANCE
Tél: (1) 44 27 71 04
Fax: (1) 44 27 62 86
{Franck.Mevel,Julien.Simon}@masi.ibp.fr
http://www-masi.ibp.fr/masix

1 Introduction

Depuis quelques années, la conception de systèmes d'exploitation répartis s'oriente vers des approches à base de micro-noyaux. En effet, ceux-ci offrent de nombreux avantages par rapport aux systèmes monolithiques, comme la portabilité, la modularité et la dynamicité. Cependant, les systèmes basés sur un micro-noyau posent de nouveaux problèmes, liés notamment à la décentralisation des informations. Afin de résoudre ces problèmes de manière efficace, le système d'exploitation doit disposer de services de communication répartis adaptés.

Masix [Card *et al.* 1993] est un système d'exploitation réparti basé sur le micro-noyau Mach [Accetta *et al.* 1986], en cours de développement au Laboratoire MASI. Après une brève présentation des services de communication offerts par Mach, de la structure de Masix et de travaux similaires aux notres, nous étudions quelques-uns des besoins du système Masix en termes de communications, à savoir la transparence, la sécurite et de bonnes performances. Puis, nous détaillons les solutions que nous avons mises au point.

2 Le micro-noyau Mach

Masix repose sur le micro-noyau Mach 3.0, développé à Carnegie Mellon University. Mach a été largement utilisé pour bâtir de nombreux systèmes compatibles avec Unix, certains centralisés comme Lites [Helander 1994] et Mach-US [Stevenson et Julin 1995], d'autres répartis comme Sprite [Kupfer 1993]. D'autres environnements ont été implémentés au-dessus de Mach, comme DOS [Malan *et al.* 1991] ou OS/2 [Phelan *et al.* 1993]. Cependant, à notre connaissance, aucun de ces systèmes n'a étudié en détail les problèmes liés au multi-environnement.

Mach offre des mécanismes puissants de communication entre tâches [Draves 1990]. Les communications sont effectuées par des ports. Un port est simplement une file dans laquelle des messages peuvent être ajoutés et retirés. Les opérations sur les ports sont effectuées via des droits sur ces ports qui sont accordés aux tâches. Trois types de droits existent :

1. droit de réception, accordé à une seule tâche ;
2. droits d'émission, accordés à plusieurs tâches ;
3. droits d'émission unique, permettant à des tâches d'envoyer un — et un seul — message sur ce port.

Chaque port est géré par une seule tâche qui possède le droit de réception sur ce port. Certains ports privilégiés sont gérés directement par le noyau Mach lui-même.

Les tâches accèdent aux ports via des noms de ports (identificateurs numériques) qui sont convertis de manière interne en droits par le noyau.

Une tâche peut envoyer ou recevoir des messages sur des ports. Un message est simplement une structure contenant des données. Un message est composé :

- d'une en-tête décrivant le message : taille du message, nom du port d'émission, nom du port de réception, type du message, code opération ;
- d'un ensemble de données typées : type de données, nombre de données, valeurs.

3 Structure du système Masix

L'objectif principal de Masix est l'exécution simultanée de plusieurs environnements, afin d'utiliser parallèlement sur une même station de travail des applications Unix, DOS, OS/2, Win32, ... Il permet également de mettre en commun les ressources d'un réseau (processeurs, mémoire, périphériques, fichiers), indépendamment des différents environnements qui peuvent s'exécuter sur chaque station.

Masix est construit selon le modèle multi-serveurs, ce qui lui confère une grande modularité. En effet, chaque serveur, qui offre des fonctionnalités orthogonales, peut être écrit et testé indépendamment des autres. De plus, il est possible d'ajouter ou de retirer dynamiquement un serveur au système.

Comme l'illustre la figure 1, Masix est composé de deux couches, elles-mêmes formées de plusieurs serveurs :

- les environnements, qui implémentent les sémantiques des systèmes d'exploitation traditionnels : Unix, DOS, OS/2, Win32, ... ;
- en dessous, le système d'accueil générique réparti, qui offre des services répartis aux environnements : communications, gestion de processus, tolérance aux fautes, ...

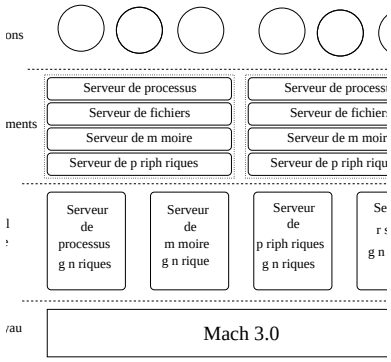

Figure 1: Structure du système Masix

4 Besoins

L'architecture multi-serveurs répartie du système Masix pose certains problèmes, comme la gestion de l'état global du système. En effet, les informations qui le composent sont réparties entre le micro-noyau et les différents serveurs. De ce fait, de nombreuses communications ont lieu, d'une part entre les différents serveurs s'exécutant sur un noeud, et d'autre part entre les différents noeuds du réseau.

Nous examinons donc trois des propriétés que doivent posséder les services de communication de Masix :

- la transparence,
- la sécurité,
- les performances.

4.1 Transparence des communications

Afin de concevoir le système Masix de manière cohérente et d'en faciliter le développement, il est préférable que les communications inter-processus distantes s'effectuent selon les mêmes sémantiques que les communications locales offertes par Mach, c'est à dire grâce à la primitive **mach_msg()**, quelle que soit la localisation des tâches.

Ainsi, deux tâches qui souhaitent communiquer n'ont pas à préjuger de leur localisation respective. Ceci est d'autant plus important que certains mécanismes de Masix, comme la migration [Mével et Simon 1995], peuvent entrainer le déplacement d'une tâche d'un noeud du réseau vers un autre noeud.

4.2 Sécurité des communications

L'architecture de Masix amplifient les problèmes traditionnels de sécurité des communications réseau. Parmi eux, les principaux sont :

- l'espionnage : il prend pour cible les liens de communications. En effet, ceux-ci à la merci de sondes physiques permettant d'intercepter l'intégralité des données qu y circulent. Si cette attaque est couronnée de succès, la confidentialité des communications est compromise. S'il est très difficile d'empêcher ce type d'attaque, il est néanmoins possible de s'en protéger grâce au chiffrement systématique des messages qui empruntent le réseau ;
- l'altération : un processus mal intentionné peut intercepter un message, le modifier et le réemettre. Par exemple, il pourrait modifier l'adresse du destinataire ou les données que le message contient. Toutefois, si le message est chiffré de manière appropriée, il ne peut pas être altéré sans que son destinataire ne s'en aperçoive. De plus, cette attaque parait difficile à mettre en place sur un réseau de type Ethernet. Elle vise plutôt les réseaux de type ¡¡ store-and-forward ¿¿ ;
- la réemission : un espion peut reémettre une séquence de messages qu'il a interceptés à destination d'un serveur. Cette attaque est délicate à contrer car l'espion n'a pas besoin de connaitre le contenu des messages. Par conséquent, le chiffrement n'est pas suffisant pour s'en protéger et il faut recourir à des méthodes d'estampillage permettant de dater les messages ;
- l'imposture : elle consiste pour un processus à obtenir des informations pour lesquelles il n'est pas habilité, en utilisant l'identité d'un processus qui, lui, l'est. Ainsi, un serveur peut être dupé par un faux client, qui accède à ses services sans autorisation et modifie son état. De même, un client peut être dupé par un faux serveur et lui transmettre des informations confidentielles. Par conséquent, il est indispensable d'authentifier les entités communicantes avant de débuter tout échange de données.

4.3 Performances

La plupart des mesures montrent que les systèmes basés sur un micro-noyau ont des performances réseau inférieures à celles des systèmes monolithiques [Maeda et Bershad 1992]. On pourrait donc en déduire que les micro-noyaux sont intrinsèquement inadaptés aux architectures réparties. Or, la plupart des systèmes basés sur le micro-noyau Mach utilisent un code réseau inadapté,

car il provient le plus souvent d'un système monolithique. C'est le cas d'Unix Server [Golub *et al.* 1990], qui utilise le code réseau de 4.3BSD.

Comme nous le verrons plus tard, il est tout à fait possible d'obtenir de bonnes performances, à condition de tenir compte des spécificités du micro-noyau.

5 Travaux existants

Nous présentons de manière succinte les principaux travaux menés à ce jour sur les communications au-dessus de Mach.

Netmsgserver

Le ¡¡ netmsgserver ¿¿ developpé par Carnegie Mellon University [Sansom *et al.* 1986] constitue la première tentative d'extension des IPC locales de Mach à l'échelle d'un réseau local.

Il est composé d'une tâche Mach multithreadée, qui s'exécute en mode utilisateur sur chaque noeud du réseau. Les serveurs réseau communiquent entre eux afin d'avoir chacun une vue cohérente de l'ensemble des tâches qui s'exécutent sur tous les noeuds.

Les principaux services assurés sont :

- conversion des ports, grâce à une base de données contenant les informations suivantes :
  - un port local, représentant la tâche locale ;
  - un port réseau, repéré par un identificateur unique, auquel sont associées certaines informations permettant de préserver la sécurité des communications ;
- gestion des ports : vérification de la validité des informations contenues dans la base de données et mise à jour si nécessaire ;
- gestion des protocoles de transport : segmentation et réassemblage des messages, contrôle de flux, gestion des erreurs de transmission ;
- gestion des messages : certaines informations contenues dans les messages Mach n'ont pas de sens à l'échelle du réseau. C'est notamment le cas des droits sur les ports. Il est donc impératif de les convertir une première fois avant de transmettre le message sur le réseau, puis à nouveau avant de livrer le message à son destinataire. Une conversion est également nécessaire lorsque l'émetteur et le destinataire n'utilisent pas la même représentation interne des données (¡¡ little-endian ¿¿ ou ¡¡ big-endian ¿¿).
- nommage ;
- chiffrement des messages, selon un algorithme dérivé de DES [NBS 1977] ;

Le ¡¡ netmsgserver ¿¿ offre donc aux tâches des services de communication transparents et sécurisés. Malheureusement, ses performances sont médiocres car chaque envoi de message nécessite de coûteux changements de contexte.

NORMA

NORMA, développé par OSF [Bryant *et al.* 1993, OSF 1994] est une extension de Mach qui permet à des tâches distantes de communiquer en utilisant les sémantiques des IPC standards de Mach. Ainsi, les communications entre tâches distantes sont totalement transparentes.

Lorsqu'une tâche envoie un droit d'émission sur un des ses ports à une tâche distante, ce port est pris en charge par NORMA : il devient ainsi un port NORMA. Chacun d'eux possède un identificateur unique. Chaque noeud gère une table de correspondance, contenant les ports NORMA dont il connait l'existence.

NORMA offre des services de communication transparents, au prix de nombreuses modifications du noyau. Toutefois, cette extension ne comporte pas à notre connaissance de services d'authentification ou de chiffrement.

6 Le serveur réseau générique

Le serveur réseau générique (SRG) offre des services de communication susceptibles de satisfaire les besoins de toutes les composantes du système Masix, c'est à dire les serveurs du système d'accueil et les serveurs qui composent les différents environnements.

6.1 Transparence des communications

Afin d'atteindre un degré total de transparence, nous étendons le modèle de communication de Mach, en interposant le SRG entre les tâches et le micro-noyau.

La figure 2 illustre la première phase du déroulement d'une communication entre deux tâches, à savoir la résolution du nom. Nous considérons deux tâches A et B, s'exécutant sur des noeuds différents, n'ayant jamais échangé de messages avec des tâches distantes, mais toutefois enregistrées auprès de leurs serveurs de noms respectifs. De plus, nous appellons R_a, R_b leurs serveurs réseau et N_a, N_b leurs serveurs de noms.

1. A fournit à N_a le nom de B et lui demande le port correspondant ;
2. comme B est une tâche distante, N_a ne connait pas ce port. Par conséquent, il demande à R_a la résolution du nom de B ;
3. R_a envoie une requête, qui contient le nom de la tâche recherchée et le numéro du noeud sur lequel A s'exécute ;
4. R_b reçoit la requête. Comme B n'a jamais communiqué avec l'exterieur, son nom lui est inconnu. R_b fournit donc à N_b le nom de B et lui demande le port correspondant ;

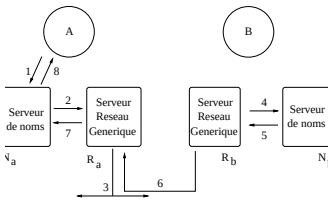

Figure 2: Résolution de nom entre deux tâches distantes

5. N_b envoie à R_b le port de B ;
6. R_b stocke cette information, puis il répond à la requête de R_a en lui envoyant le port de B et le numéro de son noeud. Ce couple (numéro de noeud, port) constitue un identificateur permettant de localiser une tâche de manière unique ;
7. R_a stocke cet identificateur et crée un port mandataire pour B. Puis, il indique à N_a le port mandataire, et lui donne un droit d'émission sur ce port ;
8. N_a indique à A le port mandataire et lui donne un droit d'émission sur ce port. Dès lors, A connait le port mandataire de B et la communication peut avoir lieu, comme l'illustre la figure 3 ;

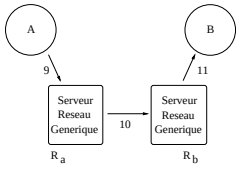

Figure 3: Communication entre deux tâches distantes

9. A envoie un message vers le port mandataire de B ;
10. comme R_a possède le droit en réception sur ce port, il reçoit le message. Il traduit les ports locaux (le port mandataire de B et le port de réponse de A) en identificateurs globaux et transmet le message à R_b.
11. R_b effectue la traduction inverse et livre le message à B.

Ce protocole de résolution de nom peut sembler terriblement coûteux. Toutefois, la résolution complète du nom de B n'a lieu qu'une seule fois sur le noeud de A. Supposons qu'une autre tâche s'exécutant sur ce noeud souhaite communiquer avec B : elle demande le port de B à N_a (étape 1), qui lui fournit immédiatement le port mandataire de B (étape 8).

Par contre, si une tâche située sur un troisième noeud souhaite communiquer avec B, la résolution complète doit avoir lieu sur ce noeud. Il est toutefois possible d'éviter toute nouvelle résolution complète en modifiant l'étape 6 : au lieu de répondre uniquement à R_a, R_b peut envoyer l'identificateur unique de B à tous les SRG du réseau. Ainsi, B est connue de tous les noeuds et la résolution de son nom se limitera à un appel au serveur de noms.

Que se passe t-il si B souhaite répondre à A ? Avant tout, le nom de A doit être résolu, de la même façon que l'a été celui de B. Cependant, la modification suivante de l'étape 10 permet de l'éviter : au lieu d'envoyer l'identificateur global de A uniquement à R_b, R_a peut l'envoyer à tous les SRG du réseau. Ainsi, A est connue de tous les noeuds.

Ces optimisations induisent des communications distantes supplémentaires. Toutefois, dans la mesure où elles remplacent un ¡¡ unicast ¿¿ par un ¡¡ broadcast ¿¿, le nombre de messages n'augmente pas. Par conséquent, nous pensons que leur impact sera faible. L'implémentation de ce protocole est en cours et seules des mesures de performances nous permettront de juger de son efficacité.

6.2 Sécurité des communications

Notre objectif est de mettre à la disposition des environnements des services permettant de déjouer les attaques que nous avons évoquées, ou du moins de compliquer leur mise en oeuvre.

Notre approche repose sur les techniques suivantes :

- séparation de l'interface de communication inter-serveurs et de l'interface de communication client-serveur ;
- chiffrement des données transmises entre les serveurs réseaux à l'aide d'un algorithme à clé publique [Diffie et Hellman 1976] ;
- authentification des serveurs par le serveur de noms :
  - lors de leur enregistrement auprès du serveur de noms à l'aide d'empreintes numériques ;
  - lors d'une demande de résolution de nom, grâce à un algorithme à clé privée ;
- authentification des serveurs entre eux, grâce à un algorithme à clé privée ;
- authentification des serveurs réseaux avant toute communication distante ;

Séparation des interfaces

Chaque serveur possède deux interfaces de communications :

- une grâce à laquelle il reçoit les requêtes des clients. Le ou les ports qui la constituent sont disponibles sur simple demande auprès du serveur de noms ;
- une lui permettant de communiquer avec d'autres serveurs. Le ou les ports qui la constituent sont également disponibles auprès du serveur de noms, mais celui-ci ne les fournit qu'après avoir authentifié son interlocuteur, c'est à dire après avoir vérifié qu'il s'agit bien d'un serveur.

Cette séparation protège les ports servant à la communication inter-serveurs et empêche ainsi un processus utilisateur d'acquérir un droit d'émission sur ces ports.

Chiffrement des transmissions entre les serveurs réseaux

Nous avons choisi d'utiliser un protocole de chiffrement à clé publique. En effet, les protocoles de ce type ne nécessitent pas la connaissance préalable d'une clé secrète de la part des entités communicantes et sont donc bien adaptés à une architecture répartie.

Le principe d'un tel protocole est très simple. Chaque entité communicante dispose de deux clés, appelées clé publique et clé secrète. Elles sont générées par un algorithme mathématique, comme RSA [Rivest *et al.* 1978] ou LUC [Smith 1993], garantissant qu'il est extrêmement difficile de déduire la clé secrète de la clé publique. La clé publique d'une entité sert à chiffrer les messages qui lui sont destinés. Par conséquent, avant de pouvoir envoyer un message à un processus, il faut obtenir sa clé publique. Une fois cette clé obtenue, le message peut être chiffré et transmis. Lorsqu'il est reçu, il est déchiffré grâce à la clé secrète, qui, commme son nom l'indique, doit rester totalement confidentielle.

Chaque serveur réseau connait la clé publique de tous les autres. En effet, lorsqu'il démarre, le SRG transmet sa clé publique à tous les autres et demandent qu'ils lui envoient la leur. Bien sûr, cet échange doit être authentifié : le fait de le chiffrer avec le mot de passe du super-utilisateur peut tenir lieu d'authentification. En effet, le seul lien entre les stations qui composent le réseau est ce mot de passe, connu seulement de l'administrateur.

Un échange sécurisé entre deux tâches A et B a lieu comme suit :

1. A envoie un message vers le port mandataire de B ;
2. comme R_a possède le droit en réception sur ce port, il reçoit le message. Il transmet le message à R_b en le chiffrant avec pub_{R_b}, la clé publique de R_b ;
3. seul R_b peut le déchiffrer grâce à sa clé secrète, sec_{R_b}. Après l'avoir fait, il transmet le message à B ;
4. lorsque B envoie la réponse à A, R_b transmet le message à R_a en le chiffrant avec pub_{R_a} ;
5. seul R_a peut le déchiffrer grâce à sec_{R_a}. Après l'avoir fait, il transmet le message à A ;
6. et ainsi de suite ...

Authentification des serveurs par le serveur de noms

Les serveurs locaux (serveurs d'environnement et serveurs génériques) s'enregistrent dès leur lancement auprès du serveur de noms. Toutefois, il faut à tout prix éviter qu'un processus utilisateur réussisse à s'enregistrer de la sorte. S'il y arrivait, il pourrait obtenir les ports de tous les serveurs, ce qui lui permettrait de consulter, voire de modifier l'état du système.

Lorsqu'il souhaite s'enregistrer, un serveur doit envoyer au serveur de noms un port, ainsi que le nom sous lequel il souhaite enregistrer ce port. Le serveur de noms demande alors au serveur de processus de déterminer le propriétaire de ce port, puis de calculer l'empreinte numérique de son exécutable, par exemple avec l'algorithme MD5 [Rivest 1992].

Si celle-ci ne figure pas dans la liste des signatures autorisées (qui peut par exemple être stockée sur un disque et chiffrée avec le mot de passe du super-utilisateur), cela signifie que le processus qui essaie de s'enregistrer n'est pas un serveur répertorié. Par conséquent, la demande d'enregistrement échoue.

Dans le cas contraire, elle réussit et le serveur de noms retourne au serveur une signature, qui fait désormais office de clé secrète. La possession de cette clé prouve que son détenteur est un serveur légitime. Grâce à cette signature, qui doit lui être fournie à chaque demande de résolution de nom, le serveur de noms peut s'assurer que son interlocuteur est un serveur enregistré, et non pas un imposteur.

Authentification entre serveurs locaux

Nous utilisons un protocole d'authentification à clé secrète, inspiré par [Needham et Schroeder 1978]. Il est illustré par la figure 4. La clé utilisée est la signature fournie par le serveur de noms après l'enregistrement du serveur.

Nous notons sec_a la clé secrète du serveur A, et $C_{sec_a}(\ldots)$ un message chiffré avec cette clé.

1. A envoie à N_a le message (A, B), c'est à dire l'identité de A et l'identité de B ;
2. N_a crée à partir de sec_a et sec_b, une clé sec_{ab} qui servira pour les communications entre A et B. Puis, il envoie à A le message $(B, sec_{ab}, C_{sec_b}(sec_{ab}, A))$;
3. A stocke sec_{ab}. Puis, il envoie à B le message $C_{sec_b}(sec_{ab}, A)$;
4. seul B est capable de déchiffrer ce message et d'obtenir sec_{ab}. Désormais, seuls A et B possèdent cette clé. Par conséquent, elle garantit l'identité de l'expéditeur.

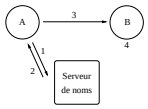

Figure 4: Authentification entre deux tâches locales

Le protocole original comporte un échange de messages supplémentaire. En effet, rien ne prouve que le message reçu par B à l'étape 4 provienne réellement de A. Il peut s'agir d'un message espionné et retransmis par un processus qui ne possède pas sec_{ab}. Cependant, étant donné qu'il s'agit de communications locales, l'espionnage ne peut pas se produire.

Authentification entre serveurs distants

Dans le cas de deux serveurs s'exécutant sur deux noeuds différents, l'obtention de la clé secrète sec_{ab} est plus compliquée, comme l'illustre la figure 5 :

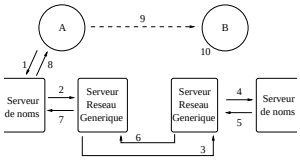

Figure 5: Authentification entre serveurs distants

1. A envoie à N_a le message (A, B) ;
2. N_a envoie à N_b le message (A, B, sec_a). Comme N_b est distante, R_a reçoit le message sur son port mandataire ;
3. R_a envoie à R_b le message $C_{pub_{R_b}}(A, B, sec_a)$;
4. R_b le déchiffre avec sec_{R_b} et envoie à N_b le message (A, B, sec_a) ;
5. N_b compose à partir de sec_a et sec_b, une clé sec_{ab} qui servira pour les communications entre A et B. Puis, il envoie à N_a le message $(A, B, sec_{ab}, C_{sec_b}(sec_{ab}, A))$. Comme N_a est distante, R_b reçoit le message sur son port mandataire ;
6. R_b envoie à R_a le message $C_{pub_{R_a}}((A, B, sec_{ab}, C_{sec_b}(sec_{ab}, A))$;
7. R_a le déchiffre avec sec_{R_a}, et envoie à N_a le message $(A, B, sec_{ab}, C_{sec_b}(sec_{ab}, A))$;
8. N_a envoie à A $(A, B, sec_{ab}, C_{sec_b}(sec_{ab}, A))$;
9. A stocke sec_{ab}. Puis, il envoie à B le message $C_{sec_b}(sec_{ab}, A)$;
10. seul B est capable de déchiffrer ce message et d'obtenir sec_{ab}. Désormais, seuls A et B possèdent cette clé. Par conséquent, elle garantit l'identité de l'expéditeur.

Ce protocole d'authentification est très proche de celui utilisé pour la résolution des noms. Par conséquent, ils peuvent sans peine être fusionnés, afin d'effectuer simultanément la localisation et l'authentification des entités communicantes.

6.3 Performances

Il est tout à fait possible d'obtenir des performances équivalentes, voire même supérieures, à celles d'un système monolithique, à condition :

- de tirer profit des spécificités du micro-noyau pour optimiser les performances :
  - pilote de périphériques ¡¡ mappé ¿¿ dans l'espace d'adressage des processus utilisateurs [Forin *et al.* 1991] ;
  - mémoire partagée entre le serveur réseau et le pilote de périphérique [Reynolds et Heller 1991] ;
- d'améliorer le fonctionnement interne du protocole, tout en préservant ses sémantiques de départ ;
- d'éviter si possible le passage des appels système liés à la communication dans un émulateur Unix en réecrivant les clients pour qu'ils utilisent directement les appels Mach. Ceci est tout à fait envisageable pour les applications courantes comme **ftp** ou **telnet**, mais plus difficile pour des applications volumineuses.

L'optimisation la plus notable est celle proposée par [Maeda et Bershad 1993]. En effet, elle permet d'atteindre les performances d'un système monolithique en décomposant les fonctionnalités d'un protocole entre le serveur réseau et une librairie ¡¡ mappée ¿¿ dans l'espace d'adressage des applications. Cette décomposition est illustrée par la figure 6.

La librairie contient les services dont les performances sont critiques, comme l'envoi ou la réception d'un message, ce qui permet d'éviter de coûteux changements de contexte entre l'application et le serveur réseau. Ce dernier

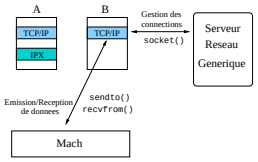

Figure 6: Décomposition des protocoles

ne contient plus quant à lui que les services qui nécessitent son intervention, comme l'ouverture et la fermeture d'une connexion. Nous travaillons sur cette décomposition.

7 Conclusion

L'architecture multi-serveurs répartie du système Masix nécessite des services de communication adaptés. Ils doivent notamment être transparents, sûrs et performants. Notre solution s'articule autour du serveur réseau générique, interposé entre les environnements et le micronoyau. Grâce à l'utilisation de ports mandataires et d'un protocole de résolution de nom, nous avons étendu les communications locales de Mach à l'échelle d'un réseau local sans modifier leurs sémantiques. Nous avons également proposé plusieurs mécanismes qui garantissent la confidentialité et l'authenticité des communications. Ils sont basés sur des algorithmes de chiffrement à clés publiques et privées. Nous avons notamment présenté un protocole d'authentification entre deux tâches distantes, qui peut très facilement être intégré dans le protocole de résolution de noms. Enfin, nous avons présenté plusieurs techniques permettant d'accroître les performances réseau de bout en bout. La plus notable est basée sur la décomposition des protocoles, afin de placer les primitives dont les performances sont critiques dans l'espace d'adressage des applications.

References

[Accetta *et al.* 1986] M. Accetta, R. Baron, W. Bolosky, D. Golub, R. Rashid, A. Tevanian, et M. Young. Mach: A New Kernel Foundation For UNIX Development. In *Proceedings of the USENIX 1986 Summer Conference*, June 1986.

[Bryant *et al.* 1993] B. Bryant, A. Langerman, S. Sears, et D. Black. *NORMA IPC: A Task-to-Task Communication System for Multicomputer Systems.* Technical report, Open Software Foundation, October 1993.

[Card *et al.* 1993] R. Card, E. Commelin, S. Dayras, et F. Mével. The MASIX Multi-Server Operating System. In *OSF Workshop on Microkernel Technology for Distributed Systems*, June 1993.

[Diffie et Hellman 1976] W. Diffie et M. Hellman. New Direction in Cryptography. *IEEE Transactions on Information Theory*, IT-22:644–654, 1976.

[Draves 1990] R. Draves. A Revised IPC Interface. In *Proceedings of the Usenix Mach Workshop*, pages 101–121, October 1990.

[Forin *et al.* 1991] A. Forin, D. Golub, et B. Bershad. An I/O System for Mach 3.0. In *Proceedings of the 1st Usenix Mach Symposium*, pages 163–176, November 1991.

[Golub *et al.* 1990] D. Golub, R. Dean, A. Forin, et R. Rashid. Unix as an Application Program. In *Proceedings of the Summer 1990 USENIX Conference*, pages 87–96, June 1990.

[Helander 1994] J. Helander. Unix under Mach - the Lites Server. Master's thesis, Helsinki University of Technology, December 1994.

[Kupfer 1993] Michael D. Kupfer. Sprite on Mach. In *Proceedings of the Third Usenix Mach Symposium*, April 1993.

[Maeda et Bershad 1992] C. Maeda et B. Bershad. Networking Performance for Microkernels. In *Proceedings of the 3rd Workshop on Workstation Operating Systems*, April 1992.

[Maeda et Bershad 1993] C. Maeda et B. Bershad. Protocol Service Decomposition for High Performance Networking. In *Proceedings of the 14th ACM Symposium on Operating Systems Principles*, pages 244–255, December 1993.

[Malan *et al.* 1991] G. Malan, R. Rashid, D. Golub, et R. Baron. DOS as a Mach 3.0 Application. In *Proceedings of the 1st Usenix Mach Symposium*, pages 27–40. Carnegie Mellon University, November 1991.

[Mével et Simon 1995] F. Mével et J. Simon. Building a Distributed Generic Layer for Multiple Personality Support on top of the Mach Microkernel. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, November 1995.

[NBS 1977] NBS. Data Encryption Standard (DES). In *Federal Information Processing Standards 46.* US National Bureau of Standards, 1977.

[Needham et Schroeder 1978] R. Needham et M. Schroeder. Using Encryption for Authentication in Large Networks of Computers. *Communications of the ACM*, 21(12):993–999, 1978.

[OSF 1994] OSF. *NORMA IPC Version Two: Architecture and Design*. Technical report, Open Software Foundation, October 1994.

[Phelan *et al.* 1993] J. Phelan, J. Arendt, et G. Ormsby. An OS/2 Personality on Mach. In *Proceedings of the 2nd Usenix Mach Symposium*, pages 191–201, April 1993.

[Reynolds et Heller 1991] F. Reynolds et J. Heller. Kernel Support For Network Protocol Servers. In *Proceedings of the Usenix Mach Symposium*, November 1991.

[Rivest *et al.* 1978] R. Rivest, A. Shamir, et L. Adelman. A method of obtaining digital signatures and public key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[Rivest 1992] R. Rivest. *The MD5 Message-Digest Algorithm*. Technical Report RFC 1321, Network Information Center, 1992.

[Sansom *et al.* 1986] R. Sansom, D. Julin, et R. Rashid. Extending a Capability Based System into a Network Environment. In *Proceedings of ACM SIGCOMM'86*, pages 265–274, 1986.

[Smith 1993] P. Smith. LUC Public Key Encryption. *Dr Dobb's Journal*, 1993.

[Stevenson et Julin 1995] J. Stevenson et D. Julin. Mach-Us: Unix on Generic OS Object Servers. In *Proceedings of the 1995 Usenix Conference*, January 1995.