INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

ADHOCFS: *A Serverless File System for Mobile Users*

David Mentré — Malika Boulkenafed — Valérie Issarny

N° 4303

Novembre 2001

THÈME 1

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--4337--FR+ENG

AdHocFS: A Serverless File System for Mobile Users

David Mentré , Malika Boulkenafed , Valérie Issarny

Thème 1 — Réseaux et systèmes
Projet SOLIDOR

Rapport de recherche n° 4303 — Novembre 2001 — 21 pages

Abstract: Among other features, pervasive computing aims at offering access to users' data, anytime, anywhere, in a transparent manner. However, realizing such a vision necessitates several improvements in the way servers and users terminals interact. In particular, users terminals should not tightly rely on an information server, which can be temporarily unavailable in a mobile situation. They should rather exploit all the information servers available in a given context through loose coupling with both fixed servers and groups of mobile terminals, in a serverless manner. This paper presents the AdHocFS serverless file system that realizes transparent adaptive file access according to the users' specific situations (e.g. device in use, network connectivity, etc).

Key-words: Distributed file system, mobile computing, security, service location protocol, wireless network.

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 Le Chesnay Cedex (France)
Téléphone : +33 1 39 63 55 11 — Télécopie : +33 1 39 63 53 30

AdHocFS: un système de fichier sans serveur pour utilisateurs nomades

Résumé : Parmi de multiples possibilités, le *pervasive computing* doit permettre l'accès aux données d'un utilisateur d'une manière transparente, en tout lieu et à tout moment. Mais mettre en œuvre une telle approche nécessite plusieurs améliorations dans la manière dont les serveurs et les terminaux utilisateurs interagissent. En particulier, les terminaux utilisateurs ne doivent pas être fortement couplés aux serveurs de données, ces derniers pouvant être temporairement indisponible dans un environnement mobile. Les terminaux doivent plutôt utiliser tous les serveurs de données à leur disposition grâce à un couplage faible avec des serveurs fixes et des groupes de terminaux mobiles, dans une approche de type « sans serveurs ». Cet article présente le système de fichier sans serveur AdHocFS qui réalise un accès aux fichiers transparent et adaptatif en fonction de la situation d'un utilisateur à un moment donné (par ex. terminal utilisé, connectivité réseau, etc.).

Mots-clés : Système de fichier distribué, informatique mobile, sécurité, protocol de localisation de service, réseaux sans fils.

1 Introduction

The vision of pervasive computing is among today's most challenging topics for information technology[1]. Realizing the vision means that consumers will be provided with universal and immediate access to available information and services, together with ways of effectively exploiting them (e.g., see [3, 6]). However, while the base networking and hardware infrastructure is here to enable the vision to become a reality, there is still a long way to go before its full realization. Open issues include provisioning: multi-modal user interfaces, software environments so that applications deployed in the computing space work effectively independently of the consumer's profile and location, network protocols for improved connectivity in any situation, and hardware for, e.g., enhanced autonomy of tiny-scale devices such as wearable computers. This paper concentrates on one such issue that is enabling users to access their data anywhere, anytime, through the provision of the AdHocFS network file system for mobile users. AdHocFS implements adaptive file access according to the users' specific situations (i.e., device in use, network connectivity, surrounding environment) without user intervention.

Offering a network file system that allows accessing files from various locations and mobile devices is not a new concern. It has been studied since the usage of laptops has become commonplace. Relevant results include work on mobile file access (e.g., [15, 10, 13, 17]) and on accessing services and data in the wide area (e.g., [5]). However, most of them concentrate on information access from mobile clients to fixed servers, possibly through an intermediate proprietary proxy. We believe that the coupling between clients and information servers should be as loose as possible in the pervasive computing space. In particular, information access should allow for information sharing within groups of mobile terminals when their location, security constraints, and the communication range of the Wireless LAN permit. An immediate benefit lies in improved performance from the standpoint of response time given the data transmission rates of wireless LAN compared to the ones of global wireless communication networks. Serverless information access has in particular been proposed in [1], which introduced xFS for fixed LAN. It is now giving rise to a number of effort in the wide area through the emergence of peer-to-peer systems (e.g., [16, 12]). Serverless information access in the context of mobile computing raises security issues since its wide acceptance will depend on how much the user can trust the system. Results from the cryptographic domain offer a number of protocols for trusted network information access. However, their usage in the mobile area has to be carefully examined given the associated computing and communication costs, which need be affordable for resource-constrained terminals. Preliminary design considerations are discussed in [9] and [18], which investigate trusted data storage on an untrusted server that is in the communication range of the mobile terminal. The issue of setting up trusted dynamic groups has also been investigated in [11]. However, implementing low-cost trusted information exchange among terminals of various capacities remains an open issue.

[1] E.g., see `www.darpa.mil/ito/Solicitations/CBD_9907.html`, `www.cordis.lu/ist/istag.htm`.

As the above discussion suggests, there exists a number of proposals that provide us with sound base ground towards offering a network file system for mobile users. Compared to existing work, our solution differs through the design of a truly mobile-aware system. The AdHocFS system provides the user with seamless access to his data, anytime, anywhere, from the terminal he currently uses, in a way that accounts for both the terminal's capacity, and content of the various local file systems of the accessible trusted mobile terminals and remote servers. The core function of AdHocFS lies in the combination of *mobile-aware naming* with *mobile-aware resource discovery* so that every file access resolves to the *nearest trusted* file location. AdHocFS further implements *mobile-aware data coherency* in a way that minimizes resource consumption while guaranteeing that the user will eventually access fresh information from the next terminal he will use. In a complementary way to the above, there is a need to account for actual data manipulation on the terminal (e.g., adapting to the size of the display). Although crucial, this issue is not considered in the remainder and we assume that the information is delivered so that the user can actually manipulate it (e.g., see [4]).

The next section discusses the design rationale of AdHocFS, addressing key requirements for our network file system together with resulting design decisions. Sections 3 to 5 then detail the key functionalities of AdHocFS when concentrating on the system support for trusted file sharing among mobile terminals; these relate to the management of security, ad hoc grouping of mobile terminals, and coherency. Section 6 presents the results of our evaluation of the AdHocFS prototype, focusing on offered response times when compared with a traditional file system aimed at a fixed LAN. Finally, Section 7 summarizes our contribution and sketches our current and future work towards the complete realization of AdHocFS.

2 Design Rationale

It is now commonplace for a person to travel with a number of personal computing terminals (e.g., laptop, handheld, mobile phone, digital camera). Currently, all these terminals are independent and transferring the data from one terminal to another requires the user's intervention, which may be more or less demanding. Then, when the user is willing to access a given personal data (e.g., retrieving a phone number, consulting his diary), he has to select the right terminal. It is further becoming common for a traveler to find various computing resources in the surrounding environment (e.g., computers available in a car, at an hotel), which may be more convenient to use in some situation compared to the traveler's personal terminal. The increasing availability of wireless networks makes possible the transfer of data from one terminal to another. Then, what needs to be achieved to make the use of the various computing resources easier for people is to enable seamless access to data, anytime, anywhere from whatever available terminal that is the most convenient in a given situation.

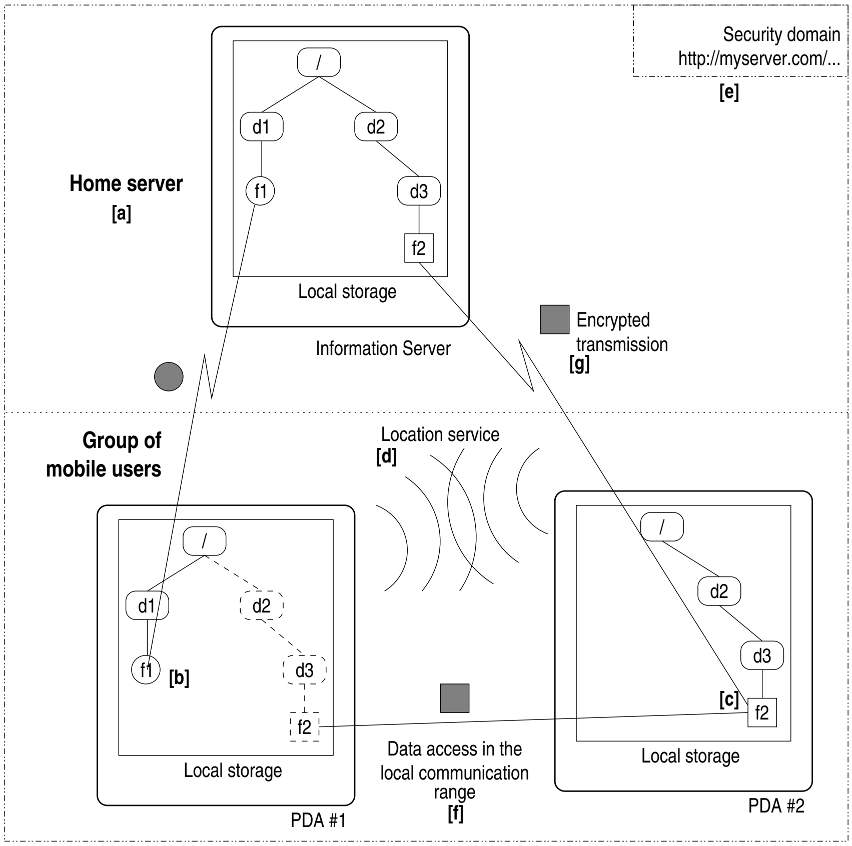

Figure 1: System overview

2.1 Offering Seamless Access

ADHOCFS is organized around a traditional file system hierarchy and the local file systems of the mobile terminals act as caches of the network file system. Then, our solution to offering seamless access to files lies in *mobile-aware naming* combined with a *mobile-aware location service*. They together allow resolving file names into the various locations from which the files may be retrieved right away.

For a given file, its locations include at least the address of the file's home server (Fig. 1-[a]) and may further extend to both a local address should the file be cached on the terminal (Fig. 1-[b]), and to terminals in the local communication range (Fig. 1-[c]) that cache the file, as identified using the mobile-aware location service. The location service serves identifying *peer mobile terminals* that are accessible by the given mobile terminal and with which files may safely be shared (Fig. 1-[d]). Such an identification relies on a traditional service discovery protocol (e.g., see [2] for a survey) combined with service identification using *security domains*. Each security domain is uniquely identified through the address of the home server storing the file system appertained to the given domain (Fig. 1-[e]). An example of security domain is the Web server (identified by its URL) hosting the information relating to a given project, which is used by the project's members to share data. Seamless access is then realized by making sure that at least the address of the user's file system stored on his home server and the ones of accessible security domains are available on all his mobile terminals, and through a component for both resolving file names according to the terminal's specifics (e.g., cached files, terminals discovered in the local communication range) and getting the files upon request. The local file system of a mobile terminal is then updated as follows. When access to a file whose name resolves only into the file's home server, is requested, a connection is established using either the wireless LAN or global wireless LAN, depending on the terminal location. The hierarchy of the local file system gets extended with the path that leads to the requested file (if missing) and the file is copied locally. In addition, upon discovery of peer terminals, the hierarchies but the files of the terminals' local file systems are merged so that any file accessible in the local communication range gets identified. Access to such a file from any of the mobile terminals then leads to copy the file locally -if not already cached.

In our context where the files get copied and updated in various locations as users move, it is crucial that each user always sees coherent data (i.e., at least the last version he accessed or, possibly, later versions that got subsequently modified by authorized users). This is dealt with through a *mobile-aware data coherency protocol* that lazily propagates updates. In addition, since mobile terminals may possibly have a small memory budget, files must be removed from the local file system upon fullness, which should ideally be done according to probability of future accesses. Design and integration of such a replacement policy in AdHocFS is an area for future work; it is currently handled in a simple way by removing files according to the LRU algorithm.

Dependability is another key requirements for enabling seamless access to data. In particular, the user will not rely on such a facility if he has no guarantee about the confidentiality and integrity of his data. It is thus mandatory to integrate adequate cryptographic techniques within the system so that the user's data can only be accessed by authorized users. Such techniques have to be used upon access and transfer of data both in the local (Fig. 1-[f]) and in the wide area (Fig. 1-[g]). The issue is then on the minimization of resource consumption and in particular energy, given the computation cost associated with cryptography and the potential limited autonomy of mobile terminals. We have thus chosen a solution that is based on private keys, which is less resource consuming than public keys.

2.2 System Architecture

Figure 2 gives an overview of the main components of AdHocFS, which offers a conventional file system API (i.e., the Linux API) and builds upon the local file system and network operating system. Core components of AdHocFS are the following:

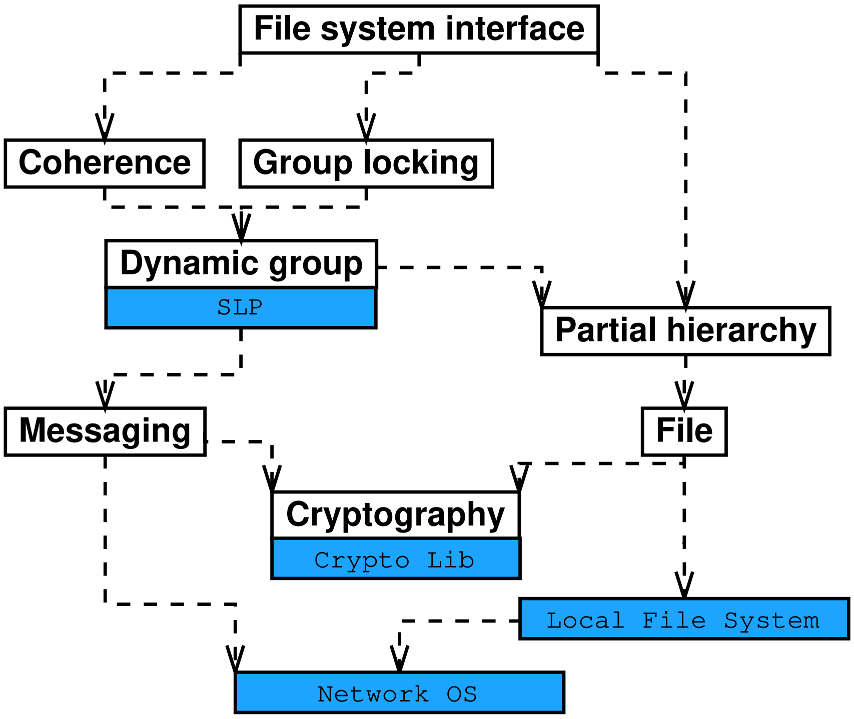

Figure 2: System class overview

- Cryptography offers functions for both symmetric and asymmetric cryptography for ensuring secure communication with file servers and peer mobile terminals.
- Messaging builds upon sockets and realizes secure message exchanges with file servers and peer mobile terminals.
- Dynamic group builds upon a base service discovery protocol and manages ad hoc grouping of peer mobile terminals, including update of the local file systems hierarchies for enabling file sharing within groups.
- Coherence manages file coherency. In particular, it deals with update propagation within ad hoc groups of mobile terminals while accounting for the groups' dynamics.
- Group locking offers base locking functions within ad hoc groups for managing file coherency.
- Partial hierarchy manages the partial file system hierarchy that is currently stored on the terminal.
- File manages requests for access to a given file, dealing in particular with file encryption.

The following focuses on the design of AdHocFS regarding its support for file sharing among peer mobile terminals, addressing the management of security (§ 3), ad hoc groups (§ 4) and coherency (§ 5).

3 Security Management

Security is of crucial importance in our context since we are using both wireless LAN and global wireless networks. It is therefore mandatory to ensure end-to-end privacy and integrity of the user's data. However, as our platform aims at running on resource constrained terminals (e.g., wireless PDA), it is necessary to balance strong security enforcement with resource consumption, and in particular energy. AdHocFS uses both asymmetric and symmetric cryptography. The former is used for securing communication links between any mobile terminal and a file server. The latter is used for securing communication links between any two peer mobile terminals. We do not further elaborate on the use of asymmetric cryptography since this paper concentrates on file sharing among mobile terminals. Let us simply mention that it enables establishing a secure link between a mobile node and a fixed server hosting the file system associated with a given security domain. Symmetric cryptography enables ensuring privacy of data between two terminals using a well-known algorithm and a shared secret, i.e., a secret key. Moreover, by using symmetric cryptography in conjunction with a cryptographically secure hash function, also called one way function, it is possible to ensure data integrity [14].

As said before, in AdHocFS, files of related interests are grouped into a *security domain*. This security domain is an administrative domain and is managed by a domain administrator. To each domain is assigned a *domain key*. This secret key is shared by the domain's server and all the terminals whose users are granted access to this domain. The domain key is set up by the domain administrator and changed on a regular basis on the domain server (i.e., regular revocation of the key for avoiding its forgery). The domain key is used for both authentication and secure message exchanges among peer mobile terminals. The use of the domain key enables minimizing the computation cost associated with cryptography, as discussed below. However, the security enforced by AdHocFS is dependent upon the avoidance of the domain key forgery, which requires regular revocation of the key. In our current prototype, we have set a fixed period for key revocation. However, thorough management of revocation is an area for future work, where we need to take into account both avoiding key forgery but also avoiding the case where peer mobile terminals know distinct domain keys and are hence unable to cooperate. Finally, protection against forgery of the domain key on the terminal itself is enforced by storing the key encrypted on the terminal, and by decrypting it using the user's password (e.g., using a PAM module in Linux). Notice that this could alternatively be realized using a smartcard, for mobile terminals equipped with a smartcard device driver.

Regarding the encryption of files transferred from one mobile node to another, the file blocks are encrypted and decrypted independently of the others (Fig. 3-[a]). However, to protect against attacks that would replace blocks within messages, the header of each such

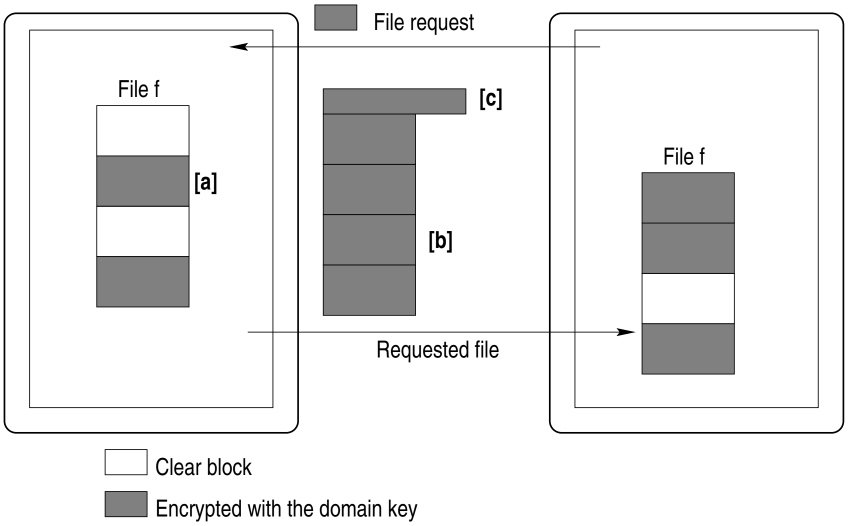

Figure 3: File encryption

message includes a secure checksum (using MD5) of the transferred blocks (Fig. 3-[c]). In addition, we use a nonce to ensure that any message received in reply to a request is indeed the reply.

Using per-block encryption allows the decryption of blocks upon actual access, instead of decryption at the time the blocks are received. In the same way, a block that is decrypted will be encrypted only if a request for the block (e.g., request for a file copy from a peer mobile terminal) is received. In this way, the computation cost associated with cryptography is minimized, and hence energy consumption due to security management is reduced. However, reduced energy consumption based on on-demand decryption is obtained due to the fact that transferred files are encrypted using the domain key (Fig. 3-[b]). If the files were encrypted using a temporary secret key that would be set up upon the establishment of a connection between any two peer mobile terminals, then upon every subsequent transmission of a file, the file would have to be decrypted (using the secret key shared with the terminal from which the file was obtained) -if not already done- and then encrypted (using the secret key shared with the requesting terminal).

4 Ad-Hoc Group Management

Ad-hoc group management in AdHocFS builds upon an existing service discovery protocol, i.e., the IETF Service Location Protocol (SLP) [8]. More precisely, we use the configuration of SLP that does not rely on a directory agent for service discovery.

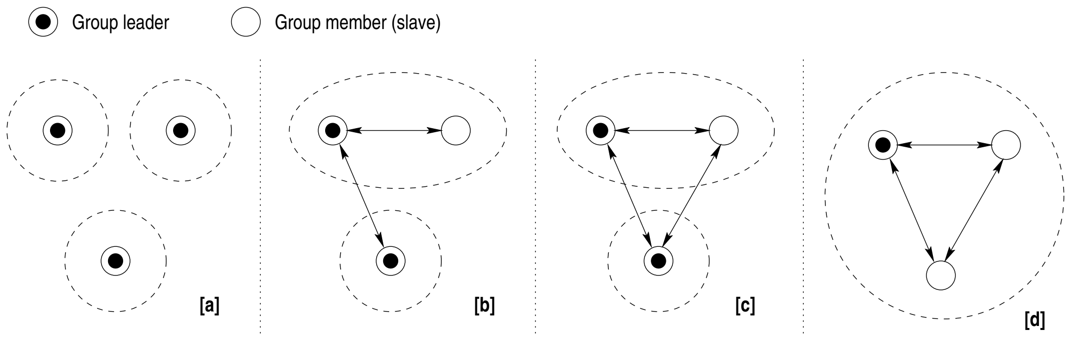

Figure 4: Merging groups

In AdHocFS, SLP serves locating peer mobile nodes, i.e., nodes in the communication range that have access to common security domains. Every mobile node acts as a *Service Agent* that handles lookup queries for peer nodes of AdHocFS. Upon initialization, each terminal registers itself for each security domain of AdHocFS it has access to, using the following format (i.e., SLP's Universal Resource Locator -URL- format [7]): service:adhocfs.inria://*IP_address*:*port_number*/*domain*[2]. Then, a mobile terminal looks for peer terminals on a regular basis, by issuing, as a *User Agent*, the following service request: service:adhocfs.inria. Every mobile terminal in the local range sends back as many messages as security domains it belongs to; each message carries the URL associated with the given security domain, which embeds the terminal's IP address, port number and domain. Using received URLs, the requesting node is able to establish secure communication links with all the peer terminals.

Once a mobile terminal has established a secure link with a peer, it invokes a specific protocol to form an ad hoc group comprising at least itself and the peer (Fig. 4). The protocol relies on a group leader that is in charge of forming the group, while the peers that are already in the group act as slaves, being informed by the leader about changes to the group. Initially, terminals are alone and hence every terminal defines a group for which it is the leader (Fig. 4-[a]). Each terminal has a Unique IDentifier (IP address and domain name in our prototype) and a unique group identifier (UID of one the terminals belonging to the group). The protocol then merges groups as they meet each other, as follows. When two leaders identify each other (Fig. 4-[b]), they first ensure that their respective group members can communicate (Fig. 4-[c]). This algorithm is iterated until all the terminals are either in the same group (Fig. 4-[d]) or cannot be joined. The latter case happens when two group leaders are in the same WLAN range but respective group members cannot reach each other (Fig. 5). Then, the two groups remain disjoint and the members of one group are kept in the *black lists* of the members of the other so as to avoid repeating the join process.

[2]Mobile terminals that do not want to cooperate (e.g., for the sake of energy saving) de-register for the security domains they belong to.

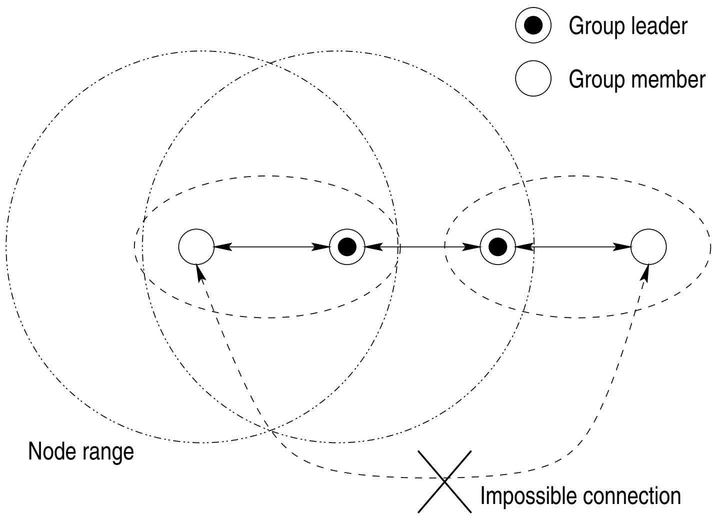

Figure 5: Partial connectivity between two groups

Once two groups are merged, a new leader is elected (the one with the smallest identifier in our prototype), and the other nodes belonging to the group, act as slave, waiting for the termination of the group initialization. The initialization of the group consists in: (i) identifying the group members, (ii) merging the file hierarchies (but the files), as stored by the leaders of the groups that merge, and (iii) broadcasting the hierarchy to the members of the composed group.

More precisely, the protocol comprises two recursive functions (Fig. 6): (i) GroupSearchAndJoin to seek a group with which to join, and (ii) GroupJoin to join a group led by a specific node k. When trying to merge with another group, which is achieved as a periodic background task, the leader i first calls SLP to locate new peer nodes (Fig. 6-[a]), i.e., nodes that respond positively to the request for the ADHOCFS service. Then, among those nodes, i selects the nodes that do not belong to the group led by i, as identified by $members_i$, nor to i's black list, as identified by $BlackList$. If the set of such peer nodes is not empty then a specific node k (the node with the smallest identifier) is chosen as being the leader of the group to merge with, and the group led by i tries to merge with the group led by k (Fig. 6-[b]). Otherwise, the protocol terminates and all the slaves are released (Fig. 6-[c]).

When trying to merge with the group led by a given node k using the GroupJoin function, the group can refuse because not all nodes of the group led by i can be reached from the members of the group led by k (Fig. 6-[d]). The node k can alternatively answer to i to retry with the group leader m (Fig. 6-[e]) or to retry later because at each step of the protocol, a node interacts with one and only one peer node (Fig. 6-[f]). In that case, a random amount of delay is inserted before retrying to merge with this group. In case of group merge, the remote peer k acknowledges the merge and gives back to i which node is the leader (Fig. 6-

let GroupSearchAndJoin$(\mathrm{BlackList}, \mathrm{members}_i) =$
[a] **let** $\mathrm{Nodes} =$ SLPLookup
if $\mathrm{Nodes} \setminus (\mathrm{BlackList} \cup \mathrm{members}_i) \neq \emptyset$ **then**
let $k = min(\mathrm{Nodes} \setminus (\mathrm{BlackList} \cup \mathrm{members}_i))$
[b] GroupJoin$(k, \mathrm{BlackList}, \mathrm{members}_i)$
else
[c] $i \rightarrow \mathrm{members}_i$: start-working$[\mathrm{gid}_i,\ \mathrm{members}_i]$
endif

let GroupJoin$(k, \mathrm{BlackList}, \mathrm{members}_i) =$
$i \rightarrow k$: group-join$[i,\ \mathrm{gid}_i,\ \mathrm{members}_i]$
match WaitAnswer **with**
[d] join-refused$[] \Rightarrow$ GroupSearchAndJoin$(\mathrm{BlackList} \cup \{k\}, \mathrm{members}_i)$
[e] join-retry-with-leader$[m] \Rightarrow$ GroupJoin$(m, \mathrm{BlackList}, \mathrm{members}_i)$
[f] join-retry-later$[] \Rightarrow$ RandomWait; GroupJoin$(k, \mathrm{BlackList}, \mathrm{members}_i)$
group-join$[j,\ \mathrm{gid}_j,\ \mathrm{members}_j]\ \wedge j \neq k \Rightarrow i \rightarrow j$: join-retry-later$[]$
[g] join-ack-leader$[\mathrm{members}_k] \Rightarrow$ GroupSearchAndJoin$(\mathrm{BlackList}, \mathrm{members}_i \cup \mathrm{members}_k)$
join-ack-slave$[\mathrm{gid}_k] \Rightarrow$ WaitAsSlave(k, gid_k)
[h] group-join$[k,\ \mathrm{gid}_k,\ \mathrm{members}_k]\ \wedge i > k \Rightarrow$ /* do not answer */
group-join$[k,\ \mathrm{gid}_k,\ \mathrm{members}_k]\ \wedge i < k \Rightarrow$
if FullyConnected$(\mathrm{members}_i, \mathrm{members}_k)$ **then**
[i] **if** $\mathrm{gid}_i < \mathrm{gid}_k$ **then**
$i \rightarrow k$: join-ack-slave$[\mathrm{gid}_i]$
GroupSearchAndJoin$(\mathrm{BlackList}, \mathrm{members}_i \cup \mathrm{members}_k)$
else
$i \rightarrow k$: join-ack-leader$[\mathrm{members}_i]$
WaitAsSlave(k, gid_k)
endif
else
[j] $i \rightarrow k$: join-refused$[]$
GroupSearchAndJoin$(\mathrm{BlackList} \cup \mathrm{members}_k, \mathrm{members}_i)$
endif
done

let WaitAsSlave$(k, \mathrm{gid}_k) =$
match WaitMessage **with**
try-connect$[j, \mathrm{members}_n] \Rightarrow$ **if** ConnectToGroup$(\mathrm{members}_n)$ **then** $l \rightarrow j$: connected$[\mathrm{members}_n]$
else $l \rightarrow j$: not-connected$[\mathrm{members}_n]$ **endif**
group-join$[n,\ \mathrm{gid}_n,\ \mathrm{members}_n] \Rightarrow l \rightarrow i$: join-retry-with-leader$[k]$
start-working$[\mathrm{gid}_i,\ \mathrm{members}_i] \Rightarrow$ **return**
done

Figure 6: Protocol for merging groups

[g]). When a node i receives a join request from k that it itself intends to merge with, only one of both nodes (the one with the smallest identifier) executes the merging algorithm (Fig. 6-[h]). If all the nodes of the group led by i can establish a connection with all the nodes of the group led by k, then both groups are merged and a new leader is chosen according to the identifiers of the respective leaders (Fig. 6-[i]). Otherwise, the nodes of the group led by k are added to i's black list and a new node (and group) is sought (Fig. 6-[j]).

When a node leaves a group, either voluntarily or not, a new group leader is chosen. It is possible to optimize this protocol by avoiding to keep a leader at all time: a new leader is elected within a group only when a group member is requested for a group merge.

5 Coherency Management

One of the main design goals for ADHOCFS is to ensure file coherency in a transparent manner. The simplest protocol to guarantee file coherency is an *exclusive writer* protocol. Using this protocol, a node can be in three modes: Read, the node can read data; ReadWrite, the node can read and write data; and Invalid, the node has no access to data. With this protocol, either a node modifies data in the ReadWrite state and all the other nodes are in the Invalid state; or either all the nodes are in the Read state. This protocol ensures that all reads access the most recent data written and thus guarantee data coherency. However, this protocol necessitates a strong connectivity with *all* remote nodes storing the data. This may not be assumed in the context of ADHOCFS where files get replicated on various mobile terminals that may happen to temporarily join a common ad hoc group, depending on their location. File coherency in ADHOCFS is managed as follows:

- The reference copy for a file is the one stored on the file's home server, and hence the version of any file is identified with respect to a timestamp that is incremented each time the reference copy is updated. Update of the reference copy occurs when a mobile terminal synchronizes with the server, which is not further discussed in the remainder. This synchronization may further lead to revocation of the domain key, in which case all the files stored on the mobile terminal are either removed from the cache or decrypted.
- File coherency within any ad hoc group is managed according to the exclusive writer protocol.
- Reconciliation of diverging copies due to non-synchronized concurrent updates (i.e., within distinct groups) is achieved using the protocol discussed hereafter.
- Diverging copies that cannot be reconciled using our protocol are ultimately combined by the user.

Although not detailed in the remainder, coherence of directories is managed in a different way, based on the semantics of the file system hierarchy. Basically, directories are merged through the union of the embedded files, where the timestamp associated with a file that is created locally on a mobile terminal is set to a special value *nil*.

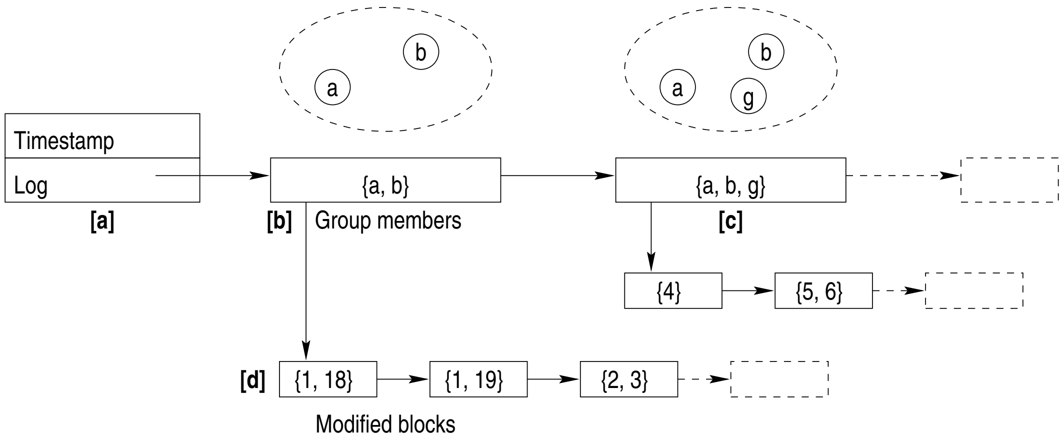

Figure 7: History attached to each file

In our protocol, a *history* is attached to each file. This history logs *all* the file updates *since* the file was copied/synchronized from/with its home server, by storing in a data structure where and when the reference file was modified[3]. An example of history is given in figure 7. The history comprises the timestamp value of the reference copy at the time it was copied/synchronized with, and the log of subsequent updates on the reference copy (Fig. 7-[a]). The log is the list of all successive groups, as seen by this specific file copy (Fig. 7-[b]). Each time the group composition is modified, through addition or deletion of a member, a new item is added to the list, which contains the new set of group members (Fig. 7-[c]), the first time a file update occurs within the given group. For each group composition, the list of the numbers of the file blocks that were modified within the group is maintained (Fig. 7-[d]). Then, reconciliation of file copies on mobile nodes is achieved by comparing the history associated with the two files.

History h_1 is a prefix of history h_2 if and only if: (i) their timestamp value of the reference copy are equal, (ii) the list of groups in h_1 is a prefix of the list of groups in h_2, and (iii) for each group composition given by the list of h_1, the list of modified blocks is the same as the one associated with the corresponding item of h_2. In addition, for the case where h_1 and h_2 have the same list of group compositions, the above condition (iii) should hold for all the group compositions but the last one, while the list of modified blocks associated with the last group composition in h_1 should be a prefix of the one of h_2. Basically, when h_1 is a prefix of h_2, this means that the file copy associated with h_1 is an earlier version than the copy associated with h_2. The older copy can then be updated by collecting the set of missing updated blocks, as identified using h_2, leading to request for these blocks to the owner of the latest file version. If two histories are not prefix of each other, then the corresponding file copies have diverged, and it is up to the user to reconcile the copies. It

[3]Notice that the number of items in the log is not a problem given the reduced size of the data structure.

is part of our future work to integrate further support for reconciliation, based on existing work in the area.

Given local histories, a mobile node is able to determine whether its local file copies are coherent with the files copies stored on peer nodes. First, when a node joins a group, a coherency check is made on all the files that are stored both locally and on peer nodes. This enables detecting diverging files as soon as possible. This is made known to the user by associating a tag to the file, which appears upon browsing. Then, it is up to the user to decide whether he wants to reconcile the file or not. Basically, divergence detection is done by comparing the histories of the various copies stored on the new comer with the ones stored within the groups[4].

We use the protocol given in Figure 8 to ensure access to a coherent copy. When reading a file f, a group read lock is taken (Fig. 8-[a]) and the identity of the latest writer in the group is stored[5]. A copy of the file is first requested if the file is not stored locally (Fig. 8-[b]). If the local file was previously in either of the ReadWrite or Read mode, or has just been copied, then the local copy is up-to-date and the read access may proceed (Fig. 8-[c]). On the other hand, if the local file was previously in the Invalid mode, then the local file copy is synchronized with the one of the writer according to history-based reconciliation (Fig. 8-[d]). Thus, when the latest writer receives the synchronize message, it compares its file copy with the one of i using their respective histories (Fig. 8-[i]). It may then reply by: (i) ok if the copies are identical, (ii) diverge if the copies diverge, or (iii) the list of encrypted blocks whose update misses on i. Finally, the new state is stored and the local read access is done. The protocol for handling write access is quite similar. First, a group write lock is taken to ensure an exclusive write access in the current group (Fig. 8-[e]), and a copy of the file is requested to the latest writer if not cached locally (Fig. 8-[f]). Then, if the file is in the Invalid state, it is possibly outdated and it is checked for coherency and possibly updated, as done in the case of read access (Fig. 8-[g]). Once the file copy is freshed, the new state is stored, the file is accessed and the set of blocks modified by the writer is added into the file's history (Fig. 8-[h]). The group write lock is finally released to allow subsequent write accesses. Updates will further be lazily propagated to the nodes storing a file copy, as accesses to the files are requested. Lazy update propagation follows from our concern of minimizing energy consumption and hence computation and communication. However, users may will to have fresh copies upon update so as to be able to later work on the latest known version of the file. This may be easily introduced in AdHocFS by associating priority-like attributes with files.

[4]For each file, the remote copy taken for comparison is the most recent file version stored in the ad hoc group, as given by the exclusive writer protocol.

[5]Locking management is realized so that a lock held by a terminal that left the group will eventually be acquired.

```
let FileRead(f) =
[a] let latest_writer = GroupReadLock(f)
[b] if not cached(f) then
      i → latest_writer: GetCopy(f.name)
      f.state ← Fresh endif
    match f.state with
[c]   Read ∨ ReadWrite ∨ Fresh ⇒ /* do nothing */
[d]   Invalid ⇒
        i → latest_writer: Synchronize(i, f.name, f.history)
      match WaitAnswer with
        ok[] ⇒ /* do nothing */
        diverge[] ⇒ raise Diverge
        update[Block] ⇒ update(f, Block)
      done
    done
    f.state ← Read
    let data = LocalRead(f)
    GroupReadUnlock(f)
    return data

let FileWrite(f, data) =
[e] let latest_writer = GroupWriteLock(f)
[f] if not cached(f) then
      i → latest_writer: GetCopy(f.name)
      f.state ← Fresh endif
    match f.state with
      Read ∨ ReadWrite ∨ Fresh ⇒ /* do nothing */
[g]   Invalid ⇒
        i → latest_writer: Synchronize(i, f.name, f.history)
      match WaitAnswer with
        ok[] ⇒ /* do nothing */
        diverge[] ⇒ raise Diverge
        update[Block] ⇒ update(f, Block)
    done
    f.state ← ReadWrite
    let modified_blocks = LocalWrite(f, data)
[h] f.history ← UpdateHistory(f.history, modified_blocks)
    GroupWriteUnlock(f)

let Synchronize(n, name, history) =
     let f = Seek(name)
     let status = compare(f.history, history)
[i]  match status with
        OK ⇒ i → n: ok[]
        Diverge ⇒ i → n: diverge[]
        Prefix ⇒
          i → n: update[missed(f.history, history)]
      done
```

Figure 8: Protocol for file access

6 Performance

A first prototype of ADHOCFS has been implemented in Objective Caml 3[6], and evaluated. The ADHOCFS prototype builds upon the Extended 2 FS (Ext2) local file system, the Blowfish symmetric encryption algorithm [19] using 128 bits keys, and the OpenSLP[7] implementation of SLP. Due to the lack of space, we focus here on a comparison with a traditional network file system (i.e., NFS), examining the respective times taken for exchanging files between two peer mobile terminals once the communication is established. In particular, we evaluate the overhead associated with cryptography. Mobile terminals of the platform are laptops with a 500 MHz Pentium III CPU, 256 KB of cache, 200 MB of RAM and a 10 GB hard disk running under Linux operating system. The wireless LAN is IEEE 802.11b in ad-hoc mode (Lucent 11Mb WaveLAN PC Card).

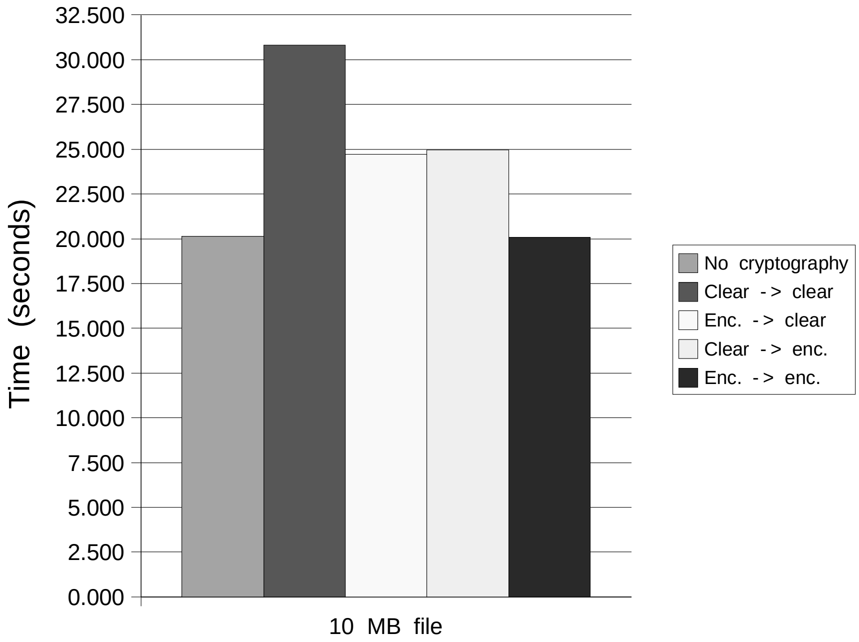

Figure 9: Cost of cryptography

Figure 9 shows the cost of cryptography for a 10 MB file transfer. In the worst case, if the file is encrypted on the sender and decrypted on the receiver, transfer time is about 1.5 slower than without any cryptography (respectively second and first bars in Fig. 9). However, when the file is encrypted or decrypted on only one node, the transfer time is only 1.2 slower (third and fourth bars). Finally, if the file is already encrypted on the sender and

[6] `caml.inria.fr`
[7] `openslp.org`

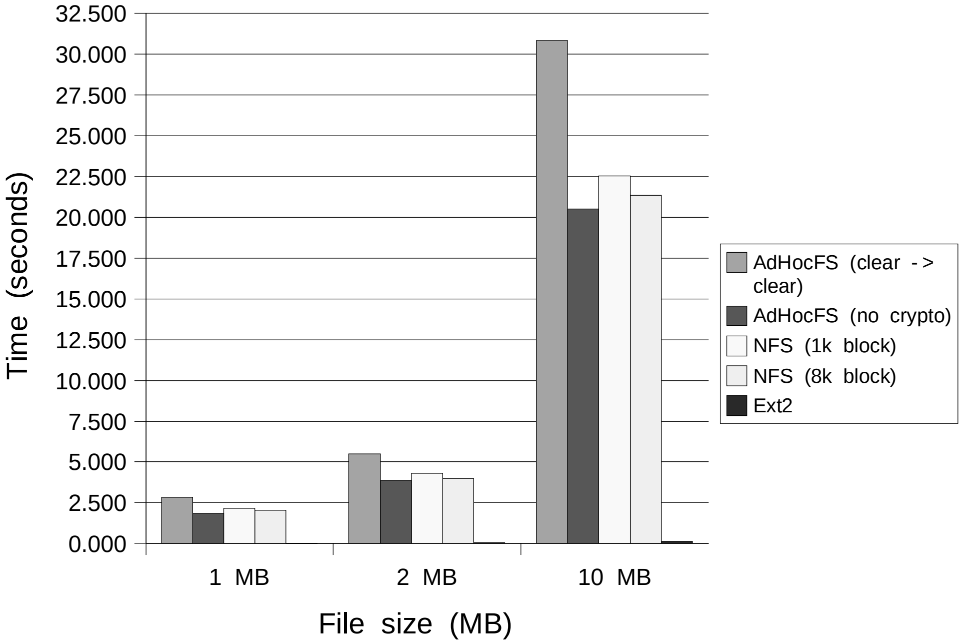

Figure 10: File transfer time

not decrypted on the receiver, transfer time is the same as without any cryptography. If the file is not decrypted on the receiver, the transfer bandwidth ranges from 3.35 (clear file on sender) to 4.20 Mbits/s (file already encrypted). Notice that the cryptographic library is written in O'Caml, which is not suited for such kind of computation. With an unoptimized C library, cryptographic overhead is divided by two.

Figure 10 presents the total transfer time for different file sizes over AdHocFS, NFS and Ext2. For each file size, the first bar shows AdHocFS file transfer time when the file is encrypted on the sender and decrypted on the receiver. The second bar presents the same transfer with disabled cryptography. The third and fourth bars are file transfers on NFS mounted with a size for transfer blocks of respectively 1k and 8k. The fifth bar (hardly visible) shows Ext2 results, i.e. local copy. When encryption is done at both ends, AdHocFS shows the slowest transfer time (by a factor of 1.5). However, when the file is just transferred between two nodes, transfer time is better than NFS, either with 1k or 8k blocks (0.9 factor). This is because, in AdHocFS, the whole file is transferred in a single message, while in NFS multiple requests should be issued, one for each block. Since a file is transferred as a whole before being used, local access time in AdHocFS is comparable to Ext2 results.

7 Conclusion

A number of network file systems for mobile users have been proposed in the past (see §1). However, most of them concentrate on information access from mobile clients to fixed servers, possibly through an intermediate proprietary proxy. We believe that the coupling between clients and information servers should be as loose as possible given today's wireless networking technology that allows accessing data available both in the local and wide area on fixed/mobile terminals. From our point of view, a network file system for mobile users must enable to: (i) seamlessly access data anywhere, anytime, in a trusted way, (ii) share information with trusted computing resources that are in the terminal's communication range, and (iii) have a coherent view of data, independent of the terminal that is being used. This paper has presented the ADHOCFS network file system for mobile users that addresses the above requirements.

In ADHOCFS, the file systems of mobile terminals act as local caches, and mobile terminals that have access to common files and are able to communicate through the wireless LAN, cooperate to form an ad hoc serverless file system. Core components of ADHOCFS lies in:

- A Mobile-aware naming scheme so that file names resolve into the various locations from which the file may be retrieved (i.e., at least the address of the file's home server, local copy if cached, peer terminals in the communication range that stores a file copy).
- A mobile-aware location service, which enables setting up ad hoc groups of mobile terminals that are connected using the wireless LAN and are granted access to common files. In addition, secure links are established between peer nodes so as to guarantee data integrity and privacy, while minimizing the computation cost and hence energy consumption associated with cryptography.
- A mobile-aware coherency management so that file copies cached on mobile terminals reconcile as terminals join in common ad hoc groups.

We have presented the results of our evaluation of the ADHOCFS prototype, focusing on offered response times when compared with a traditional file system aimed at a fixed LAN. Our results show that the overhead due to encryption of files is directly proportional to actual access to blocks.

Among enhancements of ADHOCFS that we are working on, we are going to further elaborate synchronization with the files' home servers so as to both minimize the occurrence of diverging copies, and regularly revoke the domain key used within groups to securely share files. We also need to enhance coherency management by offering support for reconciling diverging copies that may not be when using only our coherency management protocol. Finally, further evaluation of the ADHOCFS prototype is needed so as to precisely quantify the cost, and in particular the energy cost, associated with ad hoc group and coherency management. We are further interested in improving the quality of service of ADHOCFS, regarding in particular availability, and in examining the potential benefit of using ad hoc network protocols in ADHOCFS, which would enable access to files in a number of hops. Finally, we are going to experiment the use of ADHOCFS with various types of mobile terminals (e.g., iPAQ), which is quite direct given the availability of Linux for embedded platforms.

Acknowledgments We would like to make a special acknowledgment to Dr. Christophe Bidan. He always made himself available for discussions and critiques of current designs and implementation issues.

This work was partially funded by ITEA VIVIAN project (`http://www-nrc.nokia.com/Vivian/index.html`).

References

[1] T. Anderson, M. D Dahlin, J. M. Neefe, D. A. Patterson, D. S. Roselli, and R. Y. Wang. Serverless network file systems. In *Proceedings of the Fifteenth ACM SOSP*, 1995.

[2] C. Bettstetter and C. Renner. A comparison of service discovery protocols and implementation of the service location protocol. In *Proceedings Sixth EUNICE Open European Summer School: Innovative Internet Applications*, 2000.

[3] M. L. Dertouzos. The future of computing. *Scientific American*, pages 52–55, 1999.

[4] A. Fox, S. D. Gribble, and Y. Chawathe. Adapting to network and client variation using active proxies: Lessons and perspectives. *Special Issue of IEEE Personal Communications on Adaptation*, 1998.

[5] S. D. Gribble, M. Welsh, R von Behren, E. A. Brewer, D. Culler, N. Borisov, , S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R. H. Katz, Z. M. Mao, S. Ross, and B. Zhao. The Ninja architecture for robust internet-scale systems and services. *Special Issue of Computer Networks on Pervasive Computing*, 2001.

[6] R. Grimm, J. Davis, B. Hendrickson, E. Lemar, A. MacBeth, S. Swanson, T. Anderson, B. Bershad, G. Boriello, S. Gribble, and D. Wetherall. Systems directions for pervasive computing. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS)*, 2001.

[7] E. Guttman, C. Perkins, and J. Kempf. Service templates and service: Schemes. RFC 2609, June 1999.

[8] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service location protocol, version 2. RFC 2608, June 1999.

[9] C. Jensen. CryptoCache: A secure sharable file cache for roaming users. In *Proceedings of the Ninth* ACM SIGOPS *European Workshop – Beyond the PC: New Challenges for the Operating System*, 2000.

[10] A. D. Joseph, A. F. De Lespinasse, J. A. Tauber, D. K. Gifford, and M. K. Kaashoek. Rover: A toolkit for mobile information access. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, 1995.

[11] Y. Kim, A. Perrig, and G. Tsudik. Simple and fault-tolerant key agreement for dynamic collaborative groups. In *ACM CCS-7*, pages 235–244, 2000.

[12] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, and B. Zhao. Oceanstore: An architecture for global-scale persistent storage. In *Proceedings of ASPLOS*, pages 190–201, 2000.

[13] G. H. Kuenning and G. J. Popek. Automated hoarding for mobile computers. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, pages 264–275, 1997.

[14] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 4th edition edition, 1996. ISBN: 0-8493-8523-7.

[15] L. B. Mummert, M. R. Ebling, and M. Satyanarayanan. Exploiting weak connectivity for mobile file access. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 143–155, 1995.

[16] Napster. www.napster.com.

[17] K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and A. J. Demers. Flexible update propagation for weakly consistent replication. In *Proceedings of the Sixteenth ACM Symposium on Operating Systems Principles*, pages 288–301, 1997.

[18] M. Satyanarayanan. Caching trust rather than content. In *Proceedings of the Ninth* ACM SIGOPS *European Workshop – Beyond the PC: New Challenges for the Operating System*, 2000.

[19] Bruce Schneier. Description of a new variable-length key, 64-bit block cipher (blowfish). In *Fast Software Encryption, Cambridge Security Workshop Proceedings (December 1993)*, pages 191–204. Springer-Verlag, 1994.

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399