Action Concertée Incitative Sécurité et Informatique 2004

MoSAIC: Mobile System Availability Integrity and Confidentiality

June 2006
18 months Progress Report

M.-O. Killijian, M. Banâtre, C. Bryce, L. Blain, P. Couderc, L. Courtès, Y. Deswarte, D. Martin-Guillerez, R. Molva, N. Oualha, D. Powell, Y. Roudier, I. Sylvain

MoSAIC: Mobile System Availability Integrity and Confidentiality

18 months Progress Report

M.-O. Killijian[α], M. Banâtre[β], C. Bryce[β], L. Blain[α], P. Couderc[β], L. Courtès[α], Y. Deswarte[α], D. Martin-Guillerez[β], R. Molva[χ], N. Oualha[χ], D. Powell[α], Y. Roudier[χ], I. Sylvain[α]

The objective of this report is to give an overview of the MoSAIC project after 18 months, halfway through the lifetime of the project. In addition to the progress overview per se, the report includes a set of surveys we produced for the project and the other project-related papers we have published. This report is thus composed of four main parts:

1. Factual Data
2. Progress Overview
3. Surveys:
    - Communication Paradigms for Wireless Mobile Appliances
    - Cooperative Backup Mechanisms
    - Cooperation Incentives
4. Published Papers:
    - Collaborative Backup for Dependable Mobile Applications
    - Sauvegarde Coopérative entre Pairs pour Dispositifs Mobiles
    - Storage Tradeoffs in a Collaborative Backup Service for Mobile Devices
    - Sauvegarde Coopérative pour Dispositifs Mobiles
    - Increasing Data Resilience of Mobile Devices with a Collaborative Backup Service

[α] LAAS-CNRS, 7 Avenue du Colonel Roche, 31077 Toulouse cedex 4
[β] IRISA, Campus Universitaire de Beaulieu, 35042 Rennes Cedex
[χ] Eurécom, 2229 Route des Cretes, Sohia Antipolis, 06560 Valbonne

1-Factual Data

Teams

The MoSAIC (Mobile Systems Availability Integrity and Confidentiality) project is composed of three teams:

- Tolérance aux Fautes et Sûreté de Fonctionnement informatique (TSF) from LAAS-CNRS, Toulouse
- Ambiant Computing and Embedded Systems (ACES) from IRISA, Rennes
- Network Security team (NS) from Institut Eurécom, Sophia-Antipolis

Participants

As of May 1st 2006, the participants are:

- Marc-Olivier Killijian (LAAS), Chargé de Recherche CNRS, as leader of the project
- Michel Banâtre (IRISA), Directeur de Recherche INRIA
- Ciaràn Bryce (IRISA), Directeur de Recherche INRIA
- Laurent Blain (LAAS), Ingénieur de Recherche CNRS
- Paul Couderc (IRISA), Chargé de Recherche INRIA
- Ludovic Courtès (LAAS), PhD student funded by ACI S&I grant
- Yves Deswarte (LAAS), Directeur de Recherche CNRS
- Damien Martin-Guillerez (IRISA), PhD student funded by IRISA, since October 2005
- R. Molva (Eurécom), Professor,
- Nouha Oualha (Eurécom), PhD student funded by Eurécom, since November 2005
- David Powell (LAAS), Directeur de Recherche CNRS
- Matthieu Roy (LAAS), Chargé de Recherche CNRS
- Yves Roudier (Eurécom), Maître de Conférence,
- Isabelle Sylvain (LAAS), Ingénieur de Recherche CNRS,

Publications

During the first 18 months of the MoSAIC project, we published the following papers (included as part 4 of this report).

[A] M.-O. Killijian, D. Powell, M. Banâtre, P. Couderc and Y. Roudier, "Collaborative Backup for Dependable Mobile Applications [Extended Abstract] ", *in 2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing*, (Toronto, Canada), Middleware 2004 Companion, pp.146-49, ACM, 2004.

[B] L. Courtès, M.-O. Killijian, D. Powell and M. Roy, "Sauvegarde coopérative entre pairs pour dispositifs mobiles", *in Deuxièmes Journées Francophones: Mobilité et Ubiquité 2005 (UbiMob'05)*, (Grenoble, France), ACM, 2005.

[C] L. Courtès, M.-O. Killijian, D. Powell, "Storage Tradeoffs in a Collaborative Backup Service for Mobile Devices", LAAS Report 05673, LAAS-CNRS, December 2005, pp 12, accepted for publication in *6th European Dependable Computing Conference*, (Coimbra, Portugal), October 2006.

[D] L. Courtès, "Sauvegarde coopérative pour dispositifs mobiles", *in Actes du 7ème Congrès des Doctorants de l'Ecole Doctorale Systèmes (EDSYS)*, LAAS Report 06227, May 2006, Tarbes, France.

[E] Damien Martin-Guillerez, "Increasing Data Resilience of Mobile Devices with a Collaborative Backup Service", in *Supplement to Proc. of the International Conference on Dependable Systems and Networks* (DSN-2006), Sheraton Society Hill, Philadelphia, PA, USA, June 2006.

2- Progress Report

Context

The MoSAIC (Mobile System Availability, Integrity and Confidentiality) project investigates novel dependability and security mechanisms for mobile wireless devices, especially personal mobile devices, in ambient intelligence applications. The mobile devices of interest include, for instance: personal digital assistants (PDAs), laptop computers, mobile telephones, digital cameras, etc., and extend to systems embedded within vehicles. The focus is on sparse ephemeral self-organizing networks, using predominately single-hop wireless communication, i.e., networks of a small number of a potentially large population of mobile devices that come into existence spontaneously by virtue of physical proximity and mutual discovery, and that cease to exist as soon as communication is no longer possible.

Most of the data carried on a PDA is a copy of data that is mainly produced and also stored elsewhere. For example, a PDA contact database is regularly synchronized with a desktop computer application. This reduces the impact of failure of such devices to the data that is produced directly on the device between synchronizations. However, in the case of capture devices (devices capable of acquiring data such as pictures, sound or video), large quantities of data are generated directly on the mobile device, leading to a much larger quantity of data that remains sensitive to device failure until a backup copy can be created. This highlights the need for new ways of ensuring data availability. Because the density of these devices is increasing (as mobile devices are becoming more and more popular), there is an opportunity for cooperatively backing up data by using neighborhood devices.

The main objective of our work is therefore to define an automatic data back-up and recovery service based on mutual cooperation between mobile devices with no prior trust relationships. Such a service aims to ensure availability of critical data managed by mobile devices that are particularly prone to energy depletion, physical damage, loss or theft. The basic idea is to allow a mobile device to exploit accessible peer devices to manage backups of its critical data.

Challenges

The implementation of such a service by cooperation between mobile nodes with no prior trust relationship is far from trivial since new threats are introduced:

(a) selfish devices may refuse to cooperate;
(b) backup repository devices may themselves fail or attack the confidentiality or integrity of the backup data;
(c) rogue devices may seek to deny service to peer devices by flooding them with fake backup requests; etc.

Furthermore, in this context, the "classical" threats to dependability and security are particularly severe: device lifetime and communication are severely limited by scarcity of electrical energy; use of wireless links means susceptibility to link attacks ranging from passive eavesdropping to active impersonation, message replay, and message distortion; poor physical protection of mobile devices (especially in a hostile environment) makes them susceptible to physical damage, and vulnerable to theft or subversion.

The project is thus addressing two related issues:

- Fault- and intrusion-tolerant collaborative data backup (with possible extension to checkpointing).
- Self-carried reputation and rewards for collaboration between sporadically interconnected and mutually suspicious peer devices without reliance on a fixed infrastructure and access to trusted third parties.

Common to both is the emphasis on spontaneous interaction between peer mobile devices with no prior trust relationships.

Our approach is very similar to that of the widely used peer-to-peer file sharing [1] and backup systems [5] on the Internet. However, the mobile environment raises novel issues and specific assumptions. First, resources on mobile devices (energy, storage space, CPU power) are scarce. Thus, the envisioned storage mechanisms need to be efficient.
In particular, in order to limit the impact on energy consumption of running the backup service, care must be taken to use wireless communication means as little as possible given that they are a heavy drain on mobile device's energy [18]. Consequently, the amount of data to transfer to perform backups needs to be kept as low as possible. Logically, this goal is compatible with that of reducing the amount of data that needs to be stored.
Second, encounters with other participating devices are unpredictable and potentially short-lived. Thus, users cannot expect to be able to transfer large amounts of data. At the storage layer, this leads to the obligation to fragment data. Inevitably, data fragments will be disseminated among several contributors. However, the global data store consisting of all the contributors' local stores must be kept consistent, in the sense of the ACID properties of a transactional database.
Again, we assume that participants in the backup service have no a priori trust relationships. Therefore, the possibility of malicious participating devices, trying to harm individual users or the service as a whole, must be taken into account.
Obviously, a malicious contributor storing data on behalf of some user could try to break the data confidentiality. Contributors could as well try to modify the data stored. Storage mechanisms used in the backup service must address this, as will be discussed next.
A number of denial of service (DoS) attacks can be envisioned. A straightforward DoS attack is data retention: a contributor either refuses to send data back to their owner when requested or simply claims to store them without actually doing so. DoS attacks targeting the system as a whole include flooding (i.e., purposefully exhausting storage resources) and selfishness (i.e., using the service while refusing to contribute). These are well-known attacks in Internet-based peer-to-peer systems [1,5,7] that are only partly addressed in the framework of ad hoc routing in mobile networks [4]. One interesting difference in defending against DoS attacks in packet routing compared to cooperative backup is that, in the former case, observation of whether the service is delivered is almost instantaneous (routing is perturbed) while, in the latter, observation needs to be performed in the longer-run (backed up data is not available when requested). More details about these challenges can be found in publication [A].

Work Carried Out

At LAAS, Ludovic Courtès began a PhD funded by the ACI S&I at the very beginning of the MoSAIC project (October 2004). His concerns, and more generally the LAAS ones, have been mostly focused on the design of the storage layer of the collaborative backup service.
At IRISA, Damien Martin-Guillerez, funded internally by IRISA, began in October 2005 a PhD focused on evaluation, simulation and resource consumption.
At Institut Eurécom, Nouha Oualha, funded internally by Eurécom, began her PhD in November 2005. Her concerns are mainly cooperation incentives and trust management.
In part 3 of this report, we provide surveys of the related state-of-the-art in the areas being addressed by the project :

- Communication Paradigms for Wireless Mobile Appliances
- Cooperative Backup Mechanisms
- Cooperation Incentives

The storage-related concerns we described above lie at the crossroads of different research domains, namely: distributed peer-to-peer backup [5,12], peer-to-peer file sharing [9], archival [16], and revision control [13,17]. Of course, each of these domains has its own primary criteria but all of them tend to share similar techniques. File sharing, for instance, uses data fragmentation to ease data dissemination and replication across the network. The other domains insist on data compression, notably inter-version compression, in order to optimize storage usage.

Study of the literature in these domains has allowed us to identify algorithms and techniques valuable in our context. These include fragmentation algorithms, data compression techniques, naming of fragments, and maintenance of suitable meta-data describing how an input stream may be recovered from fragments.

We implemented some of these algorithms and evaluated them in different configurations. Our main criteria were storage efficiency and computational cost. We performed this evaluation using various classes of data types that we considered representative of what may be stored on typical mobile devices. Those studies and the results are available in publications [B] and [C].

As mentioned earlier, backup replication must be done as efficiently as possible. Commonly, erasure codes [20] have been used as a means to optimize storage efficiency for a desired level of data redundancy. Roughly, erasure codes produce n distinct symbols from a k-symbol input, with $n > k$; any $k + \varepsilon$ symbols out of the n output symbols suffice to recover the original data. Therefore, $n - (k + \varepsilon)$ erasures can be tolerated, while the effective storage usage is $\frac{k+\varepsilon}{n}$. A family of erasure codes, namely rateless codes, can potentially produce an infinity of output symbols while guaranteeing (probabilistically) that still only $k + \varepsilon$ output symbols suffice to retrieve the data [14].

However, an issue with erasure codes is that whether or not they improve overall data availability is highly dependent on the availability of each component storing an output symbol. In a RAID environment where the availability of individual drives is relatively high, erasure codes are beneficial. However, in a loosely connected scenario, such as a peer-to-peer file sharing system, where peers are only available sporadically at best, erasure codes can instead hinder data availability [2,10,19]. Therefore, as described in publication [D], we need to assess the suitability of erasure codes for our application.

Furthermore, data dissemination algorithms need to be evaluated. Indeed, several data fragment dissemination policies can be imagined. In order to cope with the uncertainty of contributor encounters, users may decide to transfer as much data as possible to each contributor encountered. On the other hand, users who give higher priority to confidentiality than to availability could decide to never give all the constituent fragments of a file to a single contributor.

We are currently in the process of analyzing these tradeoffs in dissemination and erasure coding using stochastic models. This should allow us to better understand their impact on data availability.

We are also studying trust management mechanisms to support cooperative services between mutually suspicious devices. Of particular interest are mechanisms based on reputation (for prior confidence-rating and posterior accountability) and rewards (for cooperation stimulation). In the sparse ephemeral networks considered, these mechanisms can rely neither on accessibility to trusted third parties nor on connectivity of a majority of the considered population of devices. Self-carried reputation and rewards are therefore of prime interest. In particular, we experimented the use of smart cards as personal purses for remuneration-based cooperation incitation. This approach assumes a reconciliation phase that may take place with an offline trust third parties. We developed an optimistic fair exchange protocol adapted to mobile backup. This approach contrasts to most existing approaches to mobile system security, which have mainly focused on key management and distribution (see, e.g., [8,11,22]) and on secure ad-hoc network routing (see, e.g., [3,6,15,21]). Achieving dependability and security despite accidental and malicious faults in networks of mobile devices is particularly challenging due to their intrinsic asynchrony (unreliable communication, partitioning, mobility, etc.) and the consequent absence of continuous consistency to global resources such as certification and authorization servers, system wide stable storage, a global time reference, etc.

Future Work

In this part of the report, we presented the recent advances in the project. Those advances mainly concerned the backup and storage aspects. In the future, our objectives are to make progress towards cooperation incentive schemes, evaluation of the replication techniques using stochastic models and development of a prototype. More details can be found in the surveys and papers included in the following parts of this report.

Concerning mobility management, we are investigating how to reuse our results in similar applications areas such as mobile sensor networks. We meet similar problems such as memory management, energy saving and data coherency among cooperating sensors.

With respect to evaluation of the replication techniques, we plan to build stochastic models (using Generalized Stochastic Petri Nets and Markov chains) to assess the probability of losing critical data produced on a mobile device when different (k,n) erasure codes are used. The basic stochastic processes of these models are assumed to be Poissonian with rates:

- λ: Device failure rate
- α: Rate of encountering contributor devices
- β: Rate of encountering usable Internet access points

The main measure of interest is the improvement in probability loss defined as the ratio of the probabilities of losing critical data respectively without and with MoSAIC.

Regarding trust management and cooperation stimulation, we plan to investigate the use of micropayment models as remuneration incentives to reduce the requirements for tamper-resistant hardware. In addition to evaluating the effectiveness of cooperation incentives, we plan to address trust modeling (probably stochastically) with respect to: cooperation expectations, data loss model, and data criticality.

Bibliography

[1] K. Bennet, C. Grothoff, T. Horozov, I. Patrascu, "Efficient Sharing of Encrypted Data", Proc. of the 7th Australasian Conference on Information Security and Privacy (ACISP 2002), (2384) pages 107–120, 2002.

[2] R. Bhagwan, K. Tati, Y-C. Cheng, S. Savage, G. M. Voelker, "Total Recall: System Support for Automated Availability Management", Proc. of the ACM/USENIX Symp. on Networked Systems Design and Implementation, 2004.

[3] Sonja Buchegger, Jean-Yves Le Boudec. "The Selfish Node: Increasing Routing Security in Mobile Ad Hoc Networks". IBM Research Report RR 3354, May 2001

[4] L. Buttyan, J-P. Hubaux, "Stimulating Cooperation in Self-Organizing Mobile Ad Hoc Networks", ACM/Kluwer Mobile Networks and Applications, 8(5), October 2003.

[5] L. P. Cox, B. D. Noble, "Pastiche: Making backup cheap and easy", Fifth USENIX OSDI, pages 285–298, 2002.

[6] B. Dahill, B.N. Levine, E. Royer, C. Shields. "A Secure Routing Protocol for Ad Hoc Networks". In *10th Conference on Network Protocols (ICNP)*, November 2002.

[7] S. Elnikety, M. Lillibridge, M. Burrows, "Peer-to-peer Cooperative Backup System", The USENIX FAST, 2002.

[8] A. Khalili, J. Katz and W. A. Arbaugh, “Toward Secure Key Distribution in Truly Ad-Hoc Networks”, in Proc. of Symp. on Applications and the Internet Workshops (SAINT'03 Workshops, pp. 342-46, 2003.

[9] D. Kügler, “An Analysis of GNUnet and the Implications for Anonymous, Censorship-Resistant Networks”, Proc. of the Conference on Privacy Enhancing Technologies, pages 161–176, 2003.

[10] W. K. Lin, D. M. Chiu, Y. B. Lee, ”Erasure Code Replication Revisited”, Proc. of the Fourth P2P, pages 90–97, 2004.

[11] D. Liu, P. Ning, K. Sun. "Efficient Self-Healing Group Key Distribution with Revocation Capability", In *10th ACM Conf. on Computer and Communications Security (CCS'03)*, (Washington D.C., USA), pp.231-40, 2003.

[12] B. T. Loo, A. Lamarca, G. Borriello, “Peer-To-Peer Backup for Personal Area Networks”, IRS-TR-02-015, UC Berkeley; Intel Seattle Research (USA), May 2003.

[13] T.Lord, “The GNU Arch Distributed Revision Control System”, 2005, http://www.gnu.org/software/gnu-arch/.

[14] M. Mitzenmacher, “Digital Fountains: A Survey and Look Forward”, Proc. of the IEEE Information Theory Workshop, pages 271–276, 2004.

[15] P. Papadimitratos, Z. J. Haas. "Secure Routing for Mobile Ad hoc Networks", In *SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002)*, (San Antonio, TX, USA), 2002.

[16] S. Quinlan, S. Dorward, “Venti: A new approach to archival storage”, Proc. of the First USENIX FAST, pages 89–101, 2002.

[17] D. S. Santry, M. J. Feeley, N. C. Hutchinson, A. C.Veitch, R. W.Carton, J.Ofir, “Deciding when to forget in the Elephant file system”, Proc. Of the 17th ACM SOSP, pages 110–123, 1999.

[18] M. Stemm, P. Gauthier, D. Harada, R. H. Katz, “Reducing Power Consumption of Network Interfaces in Hand-Held Devices”, IEEE Transactions on Communications, E80-B(8), August 1997, pages 1125–1131.

[19] A.Vernois, G.Utard, “Data Durability in Peer to Peer Storage Systems”, Proc. of the 4th Workshop on Global and Peer to Peer Computing, pages 90–97, 2004.

[20] L. Xu, “Hydra: A Platform for Survivable and Secure Data Storage Systems”, Proc. of the ACM Workshop on Storage Security and Survivability, pages 108–114, 2005.

[21] M.G. Zapata, N. Asokan. "Securing Ad Hoc Routing Protocols". In *ACM Workshop on Wireless Security (WiSe 2002)*, September 2002.

[22] L. Zhou and Z.J. Haas, “Securing Ad Hoc Networks”, IEEE Network Magazine vol 13(6), pp. 24-30, 1999, IEEE.

3- Surveys

A survey on communication paradigms for wireless mobile appliances

Damien Martin-Guillerez [*]and Michel Banâtre [†]and Paul Couderc[‡]

July 3, 2006

[*]ENS-Cachan/Bretagne - dmartin@irisa.fr
[†]INRIA - banatre@irisa.fr
[‡]INRIA - pcouderc@irisa.fr

Contents

1 Introduction

Wireless interfaces usually support only short-range communications. Thus, wireless appliances can only communicate with close neighbors and messages require to be retransmitted to attain the whole network. Moreover, mobility of appliances leads to regular changes of neighbors and of network capabilities.

Several proposals have been made to address these issues. Adaptive approaches concentrate on simulating classical networks by palliating disconnection, bandwidth decrease and limited communication range using quality of service and routing mechanisms. On the contrary, ubiquitous approaches address only neighbor devices and use wireless limitations and mobility as information for the services.

In this survey, section 2 presents mainstream wireless technologies and their limitations. Section 3 presents the adaptive approaches and section 4 the ubiquitous ones. Finally, after underlining the retained solutions for the MoSAIC project [14] in section 5, we conclude in section 6.

2 Wireless network technologies

IEEE 802.11 and BlueTooth are the main communication technologies that exist today. In this section, we will present these technologies and some new ones that are starting to appear.

2.1 IEEE 802.11

IEEE 802.11 [24, 8], often nicknamed WiFi, is a standard for wireless local area networks (WLAN). It was designed to remove cables in short-range local area networks. Its communication range is about 50 meters in an open office environment for a bandwidth from 10 to 54 Mbps per channel. It has two main communication modes: infrastructure and ad hoc.

In the infrastructure mode, IEEE 802.11 uses access points to act as routers between peers (and the possible wired network). In this mode, access to other terminals is similar to Ethernet. Each wireless interface has a unique identifier (its MAC address) which is used when addressing another interface. All messages pass through the access point which redirects them to the corresponding interface (if it has registered to the access point).

In the ad hoc mode, every interface uses the same network parameters can communicate directly with the others.

2.2 BlueTooth

BlueTooth [12] is the name for the IEEE 802.15.1 [4] standard for wireless personal area networks. It has been designed for energy efficient communications in very short range (less than 10 meters). It has a 1 Mbps bandwidth. Contrary to the IEEE 802.11 ad hoc mode, BlueTooth relies on a master-slave communication paradigm where each message between two peers passes through a node called the *piconet master*[1].

To enter a network, BlueTooth uses a discovery mode (Inquiry Scan) during which it can detect new nodes but the scan requires 1,28 seconds per node minimum. Moreover, during the discovery mode, the interface cannot be reached for other activities. The very slow discovery time and the invisibility during the discovery are real problems to when it comes to handling mobility. Therefore, BlueTooth is generally used for communication between fairly static sets of BlueTooth capable appliances (like a PC and a BlueTooth mouse).

2.3 Other technologies

Several other technologies have started to appear. ZigBee (IEEE 802.15.4) is a more energy and memory efficient technology than BlueTooth for close range networks. Unfortunately, it suffers

[1] A *piconet* is defined as a small BlueTooth network including a master and several slaves.

from a lower bandwidth (250 kbps) and discovery time. ZigBee is aimed mainly at networks of non-mobile sensor nodes.

WiMax or IEEE 802.16 [23] is a standard for wide area wireless networks and provides a network similar to the the infrastructure mode of IEEE 802.11. It can provide a bandwidth from 4.5 Mbps to 21 Mbps per channel in a range of 1 to 15 kilometers. Thus, it is mainly designed for a large-scale cellular service.

3 Adaptive approaches

The usual approaches to handle mobility in wireless networks are the adaptive ones. These approaches rely on implicit mechanisms to emulate a continuous connectivity in a mobile environment.

3.1 Handover handling in infrastructured networks

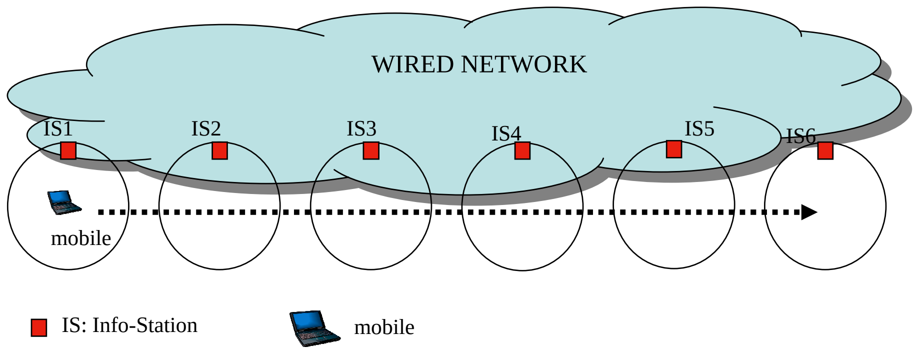

Figure 1: Infrastructured network.
Multiple linked cells (IS1 to IS6) cover the area and the mobile terminal can move between them.

In an infrastructured wireless network, each cell is covered by a wireless antenna (e.g., a HotSpot or an InfoStation [10]) like in figure 1. Mobile terminals move between those cells and thus suffer from bandwidth variations (loss of network and decrease of bandwidth). To address this issue, QoS (Quality of Service) and caching mechanisms are generally used.

QoS mechanisms decrease or increase the bit-rate of the data sent to the terminal depending on the network capabilities. For a video stream for instance, the encoding bit-rate of the video can be modified to match the network bandwidth. Caching mechanisms send data that will be of use later when the bandwidth of the network is high. In disconnected or poor bandwidth zones, the mobile terminal can work on pre-loaded data.

3.2 Ad hoc routing

In mobile ad hoc networks (MANETs) [13], each terminal can act as a router (see figure 2) like in peer to peer networks [5]. Protocols like LAR [15] or DREAM [2] use flooding like methods to construct routes. LAR sends a request to a certain area (the expected zone) when a route needs to be constructed to send a message: it is a reactive protocol. On the other hand, DREAM is proactive: it regularly reconstructs its routes by sending discovery messages in a precise zone (similar to the expected zone of LAR). The expected zones of LAR and DREAM are both computed using cinematic information although the computations are different. These *expected zones* successfully reduce the bandwidth and generally scale correctly with the mobility.

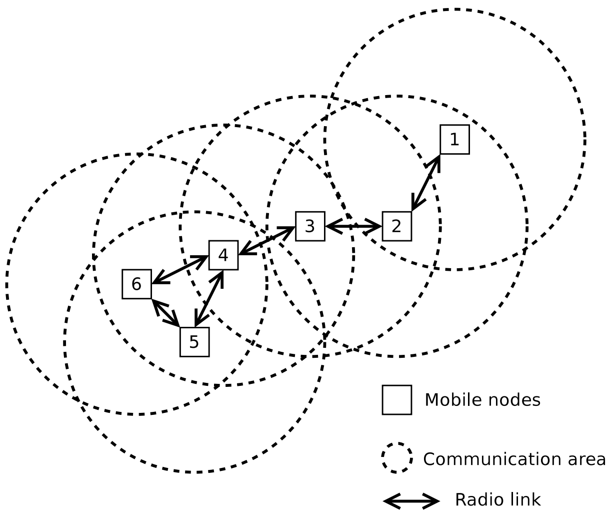

Figure 2: A Mobile Ad Hoc Network.
Each nodes can only communicate with some others (6 can only talk to 4 and 5). To talk to other nodes, messages must be re-transmitted by intermediary nodes (a message from 6 to 2 must be routed by nodes 4 and 3). The links between nodes change during the evolution of the network (nodes are mobile).

3.3 Predicting handover and change of cell

The knowledge of the way a terminal moves is a key point for handling mobility in wireless networks. If the system is able to know where and when a terminal will switch cells or will be disconnected then it can anticipate mobility by sending data to the next cell, authenticating the terminal in the next cell, etc...

Abowd et al. [1] and Narendran et al. [16] have proposed mechanisms to predict the minimum time before a terminal leaves the coverage zone. It can be done using regular measurement of the received signal power. This power is proportional to the inverse of the square of the distance ($\frac{1}{d^2}$) in short range and to $\frac{1}{d^4}$ in long range. Thus, using regular acquisitions of the received signal power, one can compute the variation of the distance and the probable time when the terminal will leave the cell. In practice, we only estimate if the signal power will be high enough during the transmission time using its measured variation. Furthermore, the IEEE 802.11F [24] standard proposes with the Inter Access Point Protocol to learn paths between different cells to predict where to send the data after the disconnection from the first cell. When a terminal moves from a cell to another, a new path is created between those two cells and data for terminals will be sent to the second cell when they start to leave the first one.

4 Ubiquitous approach

Contrary to adaptive approaches, the ubiquitous approach based on the Mark Weiser's vision [22] considers that each wireless terminal is an information system that has to be available only in close range (figure 3). For example, electronic tags provide informations on each painting of a museum and a visitor's handled computer can read those tags and so act as a virtual guide.

4.1 Spontaneous communications

The underlying concept in ubiquitous computing is spontaneous communications. As ubiquitous computing's main principle is that of invisible embedded computers that enable users to seamlessly

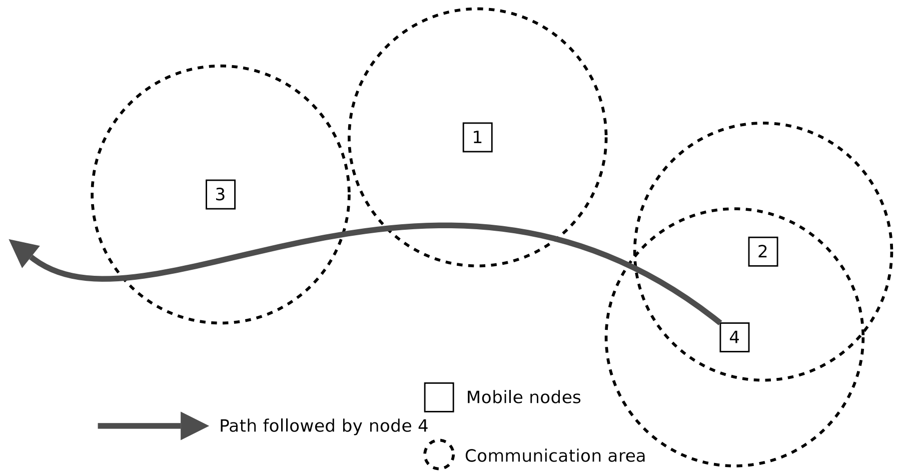

Figure 3: A ubiquitous service.
Each node provides information in its range of communication. Node 4 can currently access to the information of node 2 and will be able to access to the information of node 1 and 3 during its trip. The accessible information is related to its position.

interact with a dynamic environment. For example, CoolTown [20] associates everyday objects with wireless appliances that contain information in the HTML format. They beacon identifiers to mobile terminals. The terminal can then display the related HTML information automatically when passing near the object. In this system, locality is a way to address information.

SPREAD [6] is a middleware for ubiquitous applications. It interprets the physical space as an addressing space. Each terminal can provide information in a tuple form and accesses information by selecting tuples of a certain form (like in Linda [11]). Those tuples are accessible only to terminals in communication range. Thus, a mobile terminal can access only neighbor information. An example of an application built using SPREAD is UbiBus, which is aimed at helping visually-impaired persons. A bus equipped with a wireless appliance using UbiBus spreads a tuple indicating the line number. User appliances that are close to the bus can acquire this information. The user is then alerted when the bus arrives.

Persend [21] is another system that uses physical space as a parameter for the addressing mechanism. It proposes to establish continuous database requests that are linked to locality. Consider the example of a terminal B that publishes a list of music albums he wants to sell and a terminal A that wants to know the list of music albums for less than ten euros. A's list evolves during the time: if A goes near B then he gets the list of B's albums that are for sale for less than ten euros, if B changes the price of an album, the request is modified accordingly and if A leaves B neighborhood, then B's albums are removed from A's list.

4.2 Communication atomicity

For some applications like taxi reservation, there is a need for transactions (e.g., atomicity of communications). Unfortunately, in presence of data loss and disconnection, the atomicity of a transaction cannot be guaranteed. However, Pauty et al. [17] have proposed a protocol to approach atomicity in the context of spontaneous communications.

They propose to add a *take* operation in SPREAD which removes a tuple from available ones. This operation is based on a four-way handshake to acknowledge the transaction as shown in figure 4. This handshake can only be started in a restricted area that can be determined either by using GPS or the strength of the radio signal used to communicate. This restricted area is calculated so that the communication time will be enough for the *take* operation to be completed as shown in figure 5.

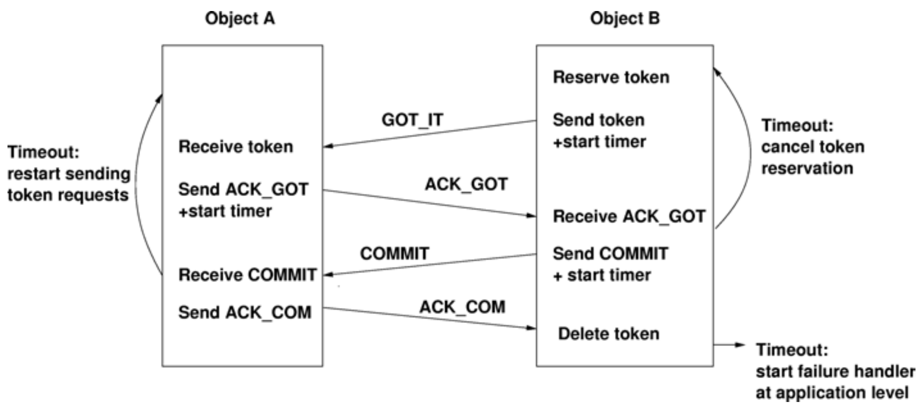

Figure 4: The *take* operation handshake.

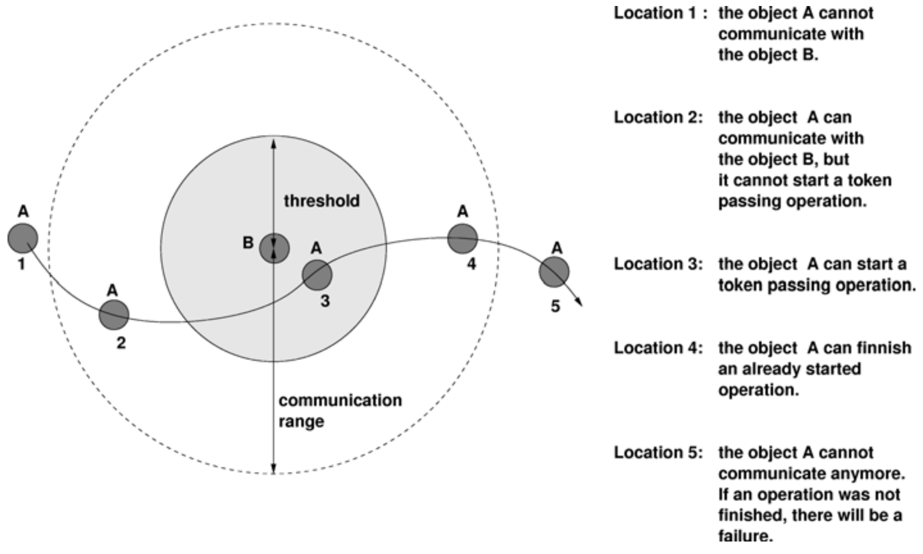
Location 1 : the object A cannot communicate with the object B.
Location 2: the object A can communicate with the object B, but it cannot start a token passing operation.
Location 3: the object A can start a token passing operation.
Location 4: the object A can finnish an already started operation.
Location 5: the object A cannot communicate anymore. If an operation was not finished, there will be a failure.
threshold
communication range
A
B
1
2
3
4
5

Figure 5: A geometric constraint guarantees a minimum communication time for the *take operation.*

Such a mechanism reduces the number of failures during atomic operations in spontaneous communications almost to null if the restricted area is small enough. The size of the ideal restricted area can be easily computed using the effective bandwidth, and the speed of change of signal (which depends on the speed of the user).

4.3 Improving resource discovery

Discovery of hosts and resources can be quite long as seen in section 2.2. In the context of high mobility, this discovery time can be a heavy burden for spontaneous communication applications. Le Bourdon et al. [3] have proposed to create *spontaneous HotSpots* that will register nearby terminals and their resources and propagate the information.

Figure 6 shows how a terminal B can transmit informations about its neighbor to a terminal A. The terminal A can then detect C without using the discovery mode of the BlueTooth interface. Of course, resources proposed by terminal C can also be given to terminal A so the latter does not have to discover them by asking C when they meet but may use them directly

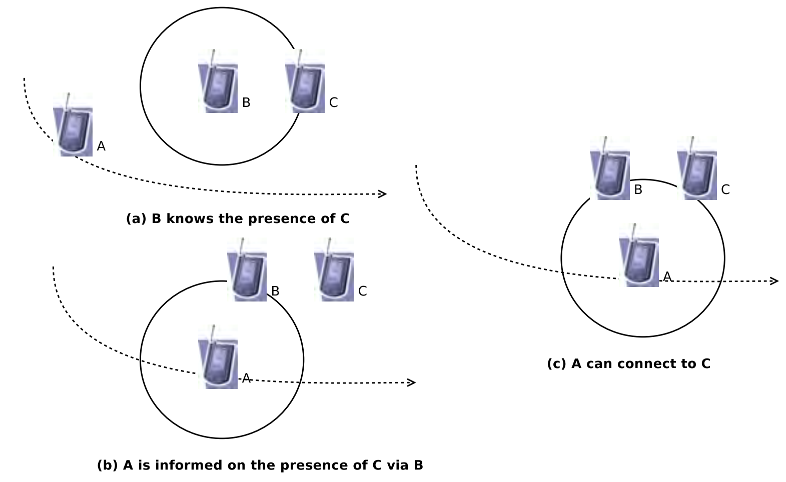
A
B
C
(a) B knows the presence of C
(b) A is informed on the presence of C via B
(c) A can connect to C

Figure 6: Propagation of resources in *spontaneous HotSpots*.

Others solutions exist to improve the discovery mode of the BlueTooth protocol but they rely on other hardware like IrDA, visual tags or dedicated devices.

5 Retained solutions for the MoSAIC project

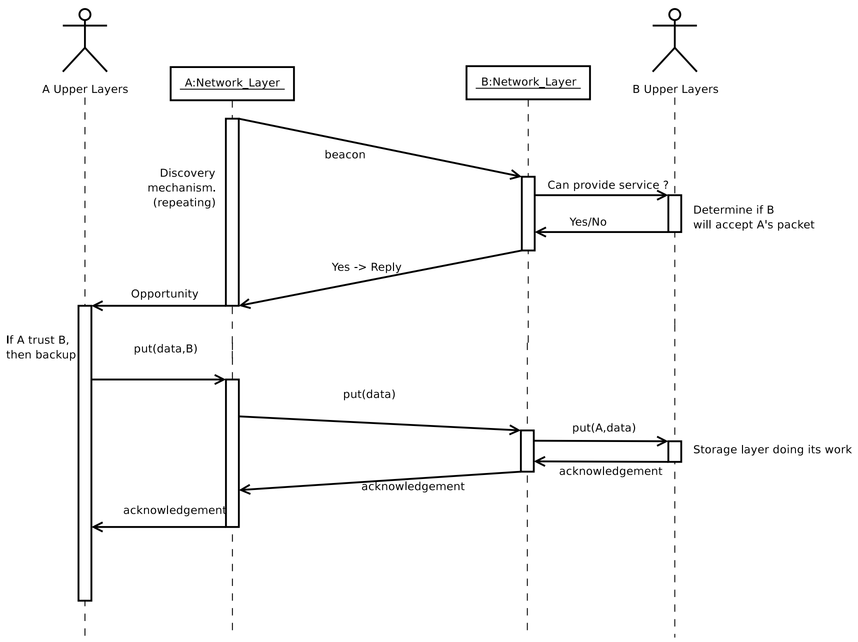

Figure 7: Interactions of the network layer in the MoSAIC project.
A terminal A tries to backup its data and a terminal B accept to save A's data.

The MoSAIC project is aiming at creating a collaborative backup for mobile appliances. Therefore, it needs a network layer providing spontaneous communications. The IEEE 802.11 standard was chosen for wireless communications because of its availability, bandwidth and communication range. The general design of MoSAIC given by Courtes et al. [7] asks for two mechanisms: a discovery and a transmission mechanisms. The figure 7 shows the required interactions for the network layer.

First, a discovery mechanism is required. It should regularly broadcast a beacon message to neighbor terminals. A neighbor terminal receiving this beacon can then acknowledge his availability for backup. The discovery mechanism receiving an acknowledgement to the beacon will signal to the other layers of MoSAIC that there is a terminal ready to save our data. The rate of sending the beacon will change depending on the amount of data to backup. Of course, the sending of the beacons will stop when all data have been backed-up. Moreover, the wireless interface can stop listening when there is no more space available for backups to avoid waste of energy. This discovery is similar to SPREAD discovery and can be done using MAC address with IPv6 [9] and UDP [18] broadcast messages.

Second, a transmission mechanism is required. The other layers of MoSAIC ask the network layer to put its data on a recently seen terminal. The network layer initiate a transmission that need to be the more atomic we can. This transmission can be done using a handshake and restricted zone like proposed by Pauty et al. However, MoSAIC only need to know if the data was correctly transmitted. Thus, TCP [19] communications can handle the required acknowledgements. The restricted zone can be determined by the power signal. Unfortunately, most IEEE 802.11 cards do not support per link statistics but give you an evaluation of the link quality of recent transmissions. So, the restricted zone will be a lower bound for the received signal power; under that bound, transmissions will be refused to avoid unnecessary energy consumption.

6 Conclusion

In this survey, we have presented the two main wireless technologies used today and their characteristics in section 2. We have seen that IEEE 802.11 is a good middle-range wireless protocol but suffers from excessive power consumption compared to BlueTooth. On the other hand, BlueTooth has a very slow discovery protocol which is a big disadvantage for spontaneous communications.

Communication paradigms for wireless mobile appliances can be divided between the adaptive and the ubiquitous approaches. Adaptive approaches try to reduce the disadvatanges of mobility by several mechanisms like caching and QoS for infrastructured networks and restricting discovery to expected zones in MANETs. Adaptive approaches also use prediction techniques using power consumption and path learning to improve data distribution or discovery. On the contrary, ubiquistic approaches use locality as a means to address data. CoolTown, SPREAD and Persend use the communication range as a way to know if the information will be useful to the user. We have also seen ways to improve spontaneous communications with atomic transactions in section 4.2 and discovery with *spontaneous hotspots* in section 4.3. Finally, we have seen several mechanisms applied to the specific case of the MoSAIC collaborative backup in section 5.

References

[1] G. D. Abowd, C. G. Atkeson, J. Hong, S. Long, R. Kooper, and M. Pinkerton. Cyberguide: a Mobile Context-Aware Tour Guide. *ACM Wireless Networks*, 3(5):421–433, Oct. 1997.

[2] S. Basagni, I. Chlamtac, R. V. Syrotiuk, and B. A. Woodward. A Distance Routing Effect Algorithm for Mobility. In *the ACM/IEEE International Conference on Mobile Computing and Networking (MOBICOM)*, pages 49–64, Oct. 1998.

[3] X. L. Bourdon, P. Couderc, and M. Banâtre. Spontaneous Hotspots: Sharing Context-Dependant Resources Using Bluetooth. In *Self-adaptability and self-management of context-aware systems (SELF'06)*, July 2006.

[4] 802.15 Specifications for wireless personnal area networks. IEEE Standards. 802.15.1 is known as BlueTooth, 802.15.4 is known as ZigBee.

[5] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *The Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, July 2000.

[6] P. Couderc and M. Banâtre. Ambient computing applications: an experience with the SPREAD approach. In *the 36th Annual Hawaii International Conference on System Sciences (HICSS'03)*, pages 291–299, Jan. 2003.

[7] L. Courtès, M.-O. Killijian, and D. Powell. Storage Tradeoffs in a Collaborative Backup Service for Mobile Devices. Technical Report 05673, LAAS-CNRS, Apr. 2006.

[8] B. P. Crow, I. Widjaja, J. G. Kim, and P. T. Sakai. IEEE 802.11 Wireless Local Area Network. *IEEE Communications Magazine*, 35(9):116–126, Sept. 1997.

[9] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) - Specification. RFC 2460, The Internet Society, Dec. 1998.

[10] R. H. Frenkiel, B. R. Badrinath, J. Borràs, and R. D. Yates. The Infostations Challenge: Balancing Cost and Ubiquity in Delivering Wireless Data. *IEEE Personal Communications*, 7(2):66–71, Apr. 2000.

[11] D. Gelernter. Generating Communication in Linda. *ACM Transactions on Programming Languages and Systems*, 7(1):80–112, Jan. 1985.

[12] J. Haartsen, M. Naghshineh, J. Inouye, O. Joeressen, and W. Allen. Bluetooth: Vision, goals, and architecture. *Mobile Computing and Communications Review*, 2(4):38–45, Oct. 1998.

[13] D. Johnson. Routing in Ad Hoc Networks of Mobile Hosts. In *The IEEE Workshop on Mobile Computing Systems and Applications (WMCSA)*, Dec. 1994.

[14] M.-O. Kilijian, D. Powell, M. Banâtre, P. Couderc, and Y. Roudier. Collaborative Backup for Dependable Mobile Applications. In *The 2nd International Workshop on Middleware for Pervasive and Ad-Hoc Computing (Middleware)*. ACM, Oct. 2004.

[15] Y.-B. Ko and N. H. Vaidya. Location-Aided Routing (LAR) in Mobile Ad Hoc Networks. *Wireless Networks*, 6(4):307–321, July 2000.

[16] B. Narendran, P. Agrawal, and D. Anvekar. Minimizing cellular handover failures without channel utilization loss. In *the Global Telecommunications Conference (GLOBECOM'94)*, volume 3, pages 1679–1685, Dec. 1994.

[17] J. Pauty, P. Couderc, and M. Banâtre. Atomic token passing in the context of spontaneous communications. Technical Report 5445, IRISA/INRIA Rennes, Jan. 2005.

[18] J. Postel. User Datagram Protocol. RFC 768, The Internet Society, Aug. 1980.

[19] J. Postel. Transmission Control Protocol. RFC 793, The Internet Society, Sept. 1981.

[20] S. Pradhan, C. Brignone, J.-H. Cui, A. McReynolds, and M. T. Smith. Websigns: Hyperlinking Physical Locations to the Web. *IEEE Computer Magazine*, 34(8):42–48, Aug. 2001.

[21] D. Touzet, F. Weis, and M. Banâtre. PERSEND: Enabling Continuous Queries in Proximate Environments. In *the Workshop on Mobile and Ubiquitous Information Access*, Sept. 2003.

[22] M. Weiser. The Computer for the Twenty-First Century. *Scientific American*, pages 94–10, Sept. 1991.

[23] 802.11 Specifications for broadband wireless access. IEEE Standards. Known as WiMax.

[24] 802.11 Specifications for wireless local area networks. IEEE Standards. Known as WiFi.

A Survey of Cooperative Backup Mechanisms [*]

Marc-Olivier Killijian Ludovic Courtès
David Powell
Laboratoire d'Analyse et d'Architecture des Systèmes
LAAS-CNRS

July 3, 2006

[*]This work is partially supported by the French Ministry of Research and Technology ACI Sécurité et Informatique MoSAIC project (http://www.laas.fr/mosaic/), the European Network of Excellence ReSIST (http://www.laas.fr/RESIST/), and the European project HIDENETS (http://www.hidenets.aau.dk/).

Contents

	File type	Writer multiplicity	Reader multiplicity
File sharing systems	Static	Single	Multiple
File backup systems	Static	Single	Single
General file systems	Dynamic	Multiple	Multiple

Figure 1: Typical characteristics distinguishing various distributed storage

1 Introduction and Motivations

Storage capacity, like computing power, follows its Moore's law and grows dramatically, for instance disk density grows at an impressive annual rate of 100% [15]. At the same time, this new storage capacity is consumed by the production of new data. Consequently, the need for backup capacity increases but so does the space available for backup.

In Section 2 of this survey, we discuss the features that characterize cooperative backup systems. Several existing systems are then described and compared briefly in Section 3. In Section 4, a more in-depth analysis is given with respect to storage management issues. Then Section 5 focuses on the dependability techniques used in these systems. Finally, Section 6 concludes the survey by a summary and sketches some directions for future work.

Our concern here is on *cooperative* backup services, in which resources belonging to multiple users are pooled as a distributed storage system for backing-up each others' data. Such a cooperative backup service must be distinguished from other forms of distributed storage such as file sharing systems or general file systems.

A file system can be defined as a support for the storage of data on non-volatile medium, typically a hard disk. Data is stored on files that encompass both the data and its associated metadata (name and other attributes such as date of modification, etc.). Usually a file system also provides a directory service on top of a flat file service. The flat file service maps the data with unique file identifiers and stores the data on the storage device. The organization of the data can have several forms, either unstructured or structured as a sequence or hierarchy of records. The directory service maps the metadata to the files' unique identifiers. Typically this mapping is stored on the storage device using the flat-file service itself.

File sharing[1] emerged relatively recently as an Internet application and greatly participated to the definition of a new type of distributed system: peer-to-peer systems. The goal of a file sharing system is to enable multiple users to access files. Classical and well-known file-sharing systems are Napster, eDonkey, Gnutella, Kazaa, MojoNation, BitTorrent, etc.

The role of a backup system is to tolerate faults affecting some storage device, be it local or distant, centralized or shared. The type of faults considered here can be permanent failures of the storage devices (e.g., crashed disks), or even localized catastrophes like a fire incident in an office when the backup media are taken off-site.

Given these definitions, one can see that there are quite some differences in the specification of these services even if there are also some strong similarities (primary goal is storage, the concept of file, etc.). If one wants more specific differences, one can consider the following properties: multiplicity of the data readers/writers and mutability of the content. The following table presents these characteristics.

Among the afore-mentioned file-sharing systems, we can identify some features that are centralized, distributed or cooperative. Similarly to file systems, file-sharing systems have two main functions: first they have to manage the actual distribution of the shared files, and second they have to organize the lookup, i.e. manage a global directory for users to search for given files. The lookup service of Napster [31] is centralized, the one of eDonkey [17] is distributed among a set of servers and finally, the one of Kademlia [28] is cooperatively realized between the participating nodes.

[1]We differentiate here between peer-to-peer file sharing systems and distributed file systems that can also be seen as a way to share files.

It is clear that file-sharing systems are different from backup systems. These systems do not guarantee long-term survivability of files, especially those files that few users are interested in storing or accessing. Thus, they could hardly be used for the purpose of backup. One could argue that regular file systems could easily be used for such a purpose since long-term survivability and fault-tolerance are very important concerns for this type of service. For instance, a simple solution to back-up on top of file systems would be to use Unix-like facilities, e.g., tar, CVS, etc. However, the specification and the semantics of file systems being so much broader than those of backup systems (multi-writers vs. single-writers; read-write vs. write-once/read-many), it would be unfair to compare these two types of system.

In this paper we will survey only backup services that use a cooperative approach. We will be concerned both with cooperation between resources pooled directly over a fixed infrastructure such as Internet and with mobile resources that are pooled opportunistically according to locality. This latter class of cooperative backup service is motivated by the observation that users of mobile devices (laptops, personal digital assistants, mobile phones, etc.) often perform a backup of their data only when they have access to their main desktop machine, by synchronizing the two machines. Typically, the first generation of personal digital assistants (PDAs) had only a short distance communication means, generally a serial or infrared device. This meant that the user had to be physically close to the machine on which she performed the synchronization. Nowadays, portable devices usually have several communication interfaces (for instance WiFi, Bluetooth, etc.). When a network infrastructure is available in their vicinity, for instance WiFi access points, those devices could connect to their main desktop machine in order to back-up their data. However, in practice, this is rarely the case, for several reasons :

- the desktop machine must be running, connected to the Internet and available;
- access to a network infrastructure using wireless communications is still rare and expensive, and it can take a while before a device is able to connect to the Internet;
- finally, to our knowledge, the software able to perform such a backup on a remote desktop are still rare.

Another solution to mobile device backup is the use of a trusted third party that guarantees its backup servers' availability. Several commercial offerings enable their customers to back-up their data on a limited storage capacity for a yearly fee.

However, the growth rate of this kind of wireless communicating devices is such that a cooperative approach to back-up is becoming feasible, based on peer-to-peer interactions. These wireless interactions are frequent but ephemeral. Nevertheless, they could be leveraged: whenever two devices meet, a backup service can automatically initiate a request for a partial data backup. As a counterpart, it has to offer the same service to the community, i.e. to form a *cooperative* backup system.

2 Characterization of Cooperative Backup Systems

In [5], Chervenak *et al.* described a number of features for the characterization of backup systems: full vs. incremental, file-based vs. device-based, on-line or not, snapshots and copy-on-write, concurrent backups, compression, file restoration, tape management, and finally disaster recovery. On the one hand, most of these features remain of interest in our context. However, on the other hand, some characteristics concerning dependability and the cooperative nature of the considered systems were not addressed: privacy, denial-of-service resilience, trustworthiness management, etc. In their survey, Chervenak et al. characterized backup systems using a set of properties. It must be noted that their focus was on centralized or server-based, system-wide, backup systems, i.e., the target was large multi-user client systems. The type of backup services we are interested in targets personal computers and thus, some of the properties defined by Chervenak et al. are not relevant. We thus propose another characterization of backup systems based on a set of functionalities and of dependability issues, as described in the following sections.

2.1 Functionalities of Backup Systems

2.1.1 Full vs. Incremental Backups

The simplest solution to back-up a file-system is to copy its entire content to a backup device. The resulting archive is called a full backup or a snapshot of the source data. Both a full file-system and a single lost file can be easily restored from such a full backup. However, this solution has usually two major drawbacks: since it concerns the entire content of the file-system, it is slow and requires a large amount of backup storage space. We will come back onto this issue of resource usage in the next subsection.

As a solution to this, incremental backup schemes can be used. They copy only the data that have been created (added) or modified since a previous backup. To restore the latest revision of a given file, the first full backup along with all the subsequent incremental backups must be read, which can be extremely slow. For this reason, typical incremental schemes perform occasional full backups supplemented with frequent incremental backups. Several variations of incremental backup techniques exist: incremental-only, full+incremental or even continuous incremental where newly created or modified data is backed-up within a few minutes, as it is created or modified, instead of once a day, typically, with traditional incremental backups.

2.1.2 Resource Usage

To reduce both storage requirements and network bandwidth usage, backup systems can use classic compression techniques. This can be done at the client-side or at the server-side. Recently, other techniques emerged to reduce the storage space required to back-up several file systems. An example is single-instance storage [33] which aims to store once every block of data even if it is present on the file systems of several users, or if there are multiple instances in a single file-system.

2.1.3 Performance

Backup system performance is measured in terms of the backup time as well as the restoration time. The performance of the backup process is impacted by factors such as incremental backups, compression, etc. Several parameters and features have a dramatic effect on the actual efficiency of the restoration operations. For instance, restoration will be slower in an incremental backup system, which must begin with the last full backup and apply changes from subsequent incremental backups. An additional concern when restoring an entire file system is that files deleted since the previous backup will reappear in the restored file system. More generally, the unbacked-up changes on the metadata, the structure and the hierarchy of the file system cannot be restored.

2.1.4 On-line Backups

While many backup systems require that the file system (or the files) remain quiescent during a backup operation, on-line or active backup systems allow users to continue accessing files during backup. On-line backup systems offer higher availability at the price of introducing consistency problems.

The most serious problems occur when directories are moved during a backup operation, changing the file system hierarchy. Other problems include file transformations, deletions and modifications during backup. In essence, any type of write operation on the files or on the file-system hierarchy during a backup is a potential source of problems. There are several possible strategies to overcome these problems:

1. *Locking* limits the availability of the system by forbidding write accesses while backing-up.
2. *Modification detection* is used to reschedule a backup of the modified structures/files.
3. *Snapshots*, i.e., frozen, read-only copies of the current state of the file-system offer another alternative for online backup. The contents of a snapshot may be backed-up without danger

of the file system being modified from subsequent accesses. The system can maintain a number of snapshots, providing access to earlier versions of files and directories.

4. A *copy-on-write* scheme is often used along with snapshots. Once a snapshot has been created, any subsequent modifications to files or directories are applied to newly created copies of the original data. Blocks or file segments are copied only if they are modified, which conserves disk space.

2.2 Dependability Issues of Cooperative Backup Systems

In the previous section we presented several functional aspects of backup systems. We now look at the dependability issued raised by a cooperative approach to backup: integrity and consistency, confidentiality and privacy, availability, synergy and trust.

2.2.1 Integrity and Consistency

A backup service has to guarantee the integrity and consistency of restored data.

Any corruption of the backed up data, be it intentional or not (for instance due to a software or hardware fault on the system actually providing the storage), must be detected by its owner during restoration. Network protocols as well as storage devices commonly use error-detecting or correcting codes to tolerate software and hardware faults. However, to be resilient to intentional corruption, the data owner must be assured that the data restored is the same that which was backed up.

Consistency is an issue when multiple items of data must ensure some common semantics. In such cases, special care must be taken to manage dependencies.

2.2.2 Confidentiality and Privacy

The entities participating in a cooperative backup service store some of their data on the resources of other participants with whom they have no a priori trust relationship. The data backed up may be private and thus should not be readable by any participating entity other than its owner, i.e., the service has to ensure the confidentiality of the data. Furthermore, a cooperative backup service must protect the privacy of its users. For instance it should not deliver any information concerning the past or present location of its users.

2.2.3 Availability

In a backup system, availability has several dimensions. First, the primary goal of a backup system is to guarantee the long-term availability of the data being backed up. In some sense it is the functional objective of the system. Second, to be useful, the backup system itself must be available, i.e., it must be resilient to failures (hardware, software, interaction, etc.). In the context of a cooperative approach to backup, additional concerns arise, especially with respect to malicious or selfish denial-of-service attacks.

2.2.4 Synergy

Synergy is the desired positive effect of cooperation, i.e., that the accrued benefits are greater than the sum of the benefits that could be achieved without cooperation. However synergy can only be achieved if nodes do indeed cooperate rather than pursuing some individual short-term strategy, i.e. being rationale.

Hardin introduced the tragedy of the commons concept in 1968 [16] to formalize the fact that a shared resource (a common) is prone to exhaustion if the resource consumers use short-term strategy to maximize their benefit out of the resource. Consider the simple example of a grass field shared by 25 farmers. The field can normally accommodate 50 cattle. However, each rational farmer is tempted to maximize his outcome by having more than 2 cattle feeding from the shared field. This short-sighted strategy eventually leads the field to exhaustion through over-consumption. A generalized form of the problem is when a resource market has externalities, i.e., when the cost of using a resource is shared among its consumers.

The tragedy of the commons has recently been extended to the digital world, or “Infosphere”, leading to the tragedy of the digital commons [13]. It is relatively intuitive, for instance, to regard the Internet as a shared resource. Each user uses his connection without paying much attention to the presence of other users and to the fact that they share a common bandwidth. Each user thus uses his available bandwidth up to its maximum, only being reminded that other users also consume this resource when there is a network congestion.

One way to ensure synergy in a cooperative backup system is to enforce the “fair exchange” property: if one contributes up to 5 MiB to the system, one wants to get serviced up to 5 MiB too. Reciprocally, it is desirable that a device getting serviced for such an amount of resources offers an equivalent amount to the cooperative service.

2.2.5 Trust Management

An important aspect of many cooperative systems is that each node has to interact with unknown nodes with which it does not have a pre-existing trust relationships. The implementation of a cooperative backup service between nodes with no prior trust relationship is far from trivial since new threats must be considered:

1. selfish devices may refuse to cooperate;

2. backup repository devices may themselves fail or attack the confidentiality or integrity of the backup data;

3. rogue devices may seek to deny service to peer devices by flooding them with fake backup requests; etc.

There is thus a need for trust management mechanisms to support cooperative services between mutually suspicious devices.

3 Existing Cooperative Backup Systems

In this section, we first give a preliminary description and analysis of various systems devoted to cooperative backup. Cooperative backup are inspired by both cooperative file systems and file sharing systems. Most are concerned with the problem of cooperative backup for fixed nodes with a permanent Internet connection. To our knowledge, there are only two projects looking at backup for portable devices with only intermittent access to the Internet: *FlashBack* [25] and *MoSAIC* [21].

3.1 Peer-to-peer Backup Systems for WANs/LANs

The earliest work describing a backup system between peers is the one of Elnikety *et al.* [11], which we will henceforth refer to as *CBS*. Regarding the functions of a backup system (resource localization, data redundancy, data restoration), this system is quite simple. First, a centralized server is used to find partners. Second, incremental backup, resource preservation, performance optimization were not addressed. However, various types of attacks against the system are described. We will come back to this later.

The *Pastiche* [8] system and its follow-up *Samsara* [9], are more complete. The resource discovery, storage, data localization mechanisms that are proposed are totally decentralized. Each newcomer chooses a set of partners based on various criteria, such as communication latency, and then deals directly with them. There are mechanisms to minimize the amount of data exchange during subsequent backups. Samsara also tries to deal with the fair exchange problem and to be resilient to denial-of-service attacks.

Other projects try to solve some limitations of the *Pastiche/Samsara* systems, or to propose some simpler alternatives. This is the case for *Venti-DHash* [38] for instance, based on the *Venti* archival system [33] of the *Plan 9* operating system. Whereas *Pastiche* selects at startup a limited set of partners, *Venti-DHash* uses a completely distributed storage among all the participants, as in a peer-to-peer file sharing system.

PeerStore [24] uses a hybrid approach to data localization and storage where each participant deals in priority with a selection of partners (like *Pastiche*). Additionally, it is able to perform incremental backup for only new or recently modified data. Finally, *pStore* [1] and ABS [6], which are inspired by versioning systems, propose a better resource usage.

Based on the observations that worms, viruses and the like can only attack machines running a given set of programs, the *Phoenix* system [20] focuses on techniques favoring diversity among software installations when backing up a machine (e.g., trying to not backup a machine that runs a given vulnerable web server on a machine that runs the same web server). The main added value is here in the partnership selection.

In [7], the authors focus on the specific issue of resource allocation in a cooperative backup system through an auction mechanism called bid trading. A local site wishing to make a backup announces how much remote space is needed, and accepts bids for how much of its own space the local site must "pay" to acquire that remote space.

In [18], the authors implement a distributed backup system, called *DIBS*, for local area networks where nodes are assumed to be trusted: the system ensures only privacy of the backed up data but does not consider malicious attacks against the service. Since *DIBS* targets LANs, all the participating nodes are known *a priori*, partnerships do not evolve, and no trust management is needed.

3.2 Cooperative File Systems

As mentioned earlier, a backup system (static data files, single writer) can be implemented on top of any file system (mutable data files, multi-writer). There exist a number of peer-to-peer general file systems such as Ivy [30], OceanStore [22], InterMemory [12], Us [34], etc. We briefly present here two of them for the sake of the comparison although they are outside the scope of this survey.

Us [34] provides a virtual hard drive: using a peer-to-peer architecture, it offers a read-only data block storage interface. On top of *Us*, *UsFs* builds a virtual file system interface able to provide a cooperative backup service. However, as a full-blown filesystem, *UsFs* provides more facilities than a simple backup service. In particular, it must manage concurrent write access, which is much more difficult to implement in an efficient way.

OceanStore [22] is a large project where data is stored on a set of untrusted cooperative servers which are supposed to have a long survival time and high speed connection. In this sense we consider it as a distributed file system using a super-peers approach rather than a purely cooperative system. The notion of super-peers relates to the fact that peers are specifically configured as file servers (with large amount of storage) that can cooperate to provide a resilient service to non-peer clients.

3.3 Mobile Systems

The *FlashBack* [25] cooperative backup system targets the backup of mobile devices in a Personal Area Network (PAN). The nature of a PAN simplifies several issues. First, the partnerships can be defined statically as the membership in the network changes rarely: the devices taking part in the network are those that the users wear or carry. Second, all the devices participating in the cooperative backup know each other. They can be initialized altogether at configuration time so there is no problem of handling dynamic trust between them. For instance, they may share a cryptographic key.

MoSAIC [21] is a cooperative backup system for communicating mobile devices. Mobility introduces a number of challenges to the cooperative backup problem. In the context of mobile devices interacting spontaneously, connections are by definition short-lived, unpredictable, and very variable in bandwidth and reliability. Worse than that, a pair of peers may spontaneously encounter and start exchanging data at one point in time and then never meet again. Unlike *FlashBack*, the service has to be functional even in the presence of mutually suspicious device users.

4 Storage Management

In this section, we present two aspects that are specific to peer-to-peer data storage systems: mechanisms for storage allocation, and techniques for efficient usage of resources.

4.1 Storage Allocation

Among the systems studied, one can identify three distinct approaches to the dissemination of the data blocks to be stored:

- the storage can be allocated to specific sets of participants or partners;
- the storage can be allocated across all participants using a distributed hash table (DHT), which has the property of ensuring an homogeneous block distribution;
- the storage can be allocated opportunistically among neighbors met when storage of a block is needed.

In the first case, the relationships between the partners are relatively simple: each participant chooses a set of partners at start-up. Then, for each backup, it directly sends the blocks to be saved to its partners. In Pastiche and in CBS, each participant chooses a set of partners that will remain almost static. Finally, the FlashBack devices, in a PAN, choose their set of partners according to the amount of time spent in each other's vicinity.

The second approach is based on a technique that is fundamental to peer-to-peer file sharing systems, *virtual networks* or *overlay networks* [27], which use the notion of distributed hash tables (DHT) for allocating data blocks. Each node of the network is responsible for the storage of the blocks whose identifier is close (numerically) to its own identifier. The advantage of using a DHT is that the blocks are homogeneously distributed over the network if their identifiers are numerically homogeneously distributed. Both Venti-DHash and pStore use DHTs to store backup data blocks. However, there are two disadvantages to this approach:

- The cost of migration of the data blocks when a node enters or leaves the system can be high (bandwidth-wise) [24]. Because of the mathematical mapping between data blocks and node identifiers, no exception is acceptable: when a node enters the virtual network, it must obtain and store all the blocks for which it is mathematically responsible; respectively, when a node leaves the network, the various blocks it was responsible for must be re-distributed using the DHT mechanism.
- A DHT automatically distributes the data blocks homogeneously among the participants, independently of how much storage space each node consumes. Consequently, using a DHT makes it impossible for a system to ensure fair exchange.

For these reasons, PeerStore proposes a hybrid approach where the data blocks are directly exchanged between partners and where the blocks' meta-information (the mapping between a block ID and the node that stores it) are stored using a DHT. For optimization, the set of partners is sometimes extended at runtime to nodes that were not originally in the partnership: when a node needs to store a block, it looks into the DHT to see if the block is already stored (single-instance storage). When that is the case, the block is not stored twice. Instead, the node that already stores it becomes a new partner for the node owning it.

The third approach is very different. The MoSAIC system targets mobile devices, so partnerships cannot be established a priori[2], but have to be defined during the backup itself, opportunistically. MoSAIC is an active backup system - whenever some critical data is modified, the modified blocks need to be backed-up. This is done towards the devices that the user will meet along its way. In this case, the partnership is determined at runtime and is a function of the mobility patterns of the participating nodes.

4.2 Storage Optimization

The amount of storage necessary to store backed-up data can be optimized by applying compression techniques. Compression is worthwhile even if data is ultimately backed-up in redundant copies (to ensure backup availability). Indeed, the redundancy that is eliminated using compression techniques can be seen as "accidental", e.g., due to overly prolix data formats. Thus, compression can be thought of as a way to *normalize data entropy* before adding new redundancy. In other words, going through the compression step before adding redundancy is a means to achieve *controlled redundancy*. In particular, controlled redundancy means that the backup software is able to control the distribution of redundant data.

Backup systems often rely on "traditional" stream compression techniques, such as *gzip* and similar tools. Additionally, most backup systems have focused on techniques allowing for storage and bandwidth savings when only part of the data of interest has been modified, i.e., incremental backup techniques. Of course, similar techniques are used by revision control systems [26] or network file systems [29].

Incremental backup has the inherent property of reducing storage (and bandwidth) usage because only changes need to be sent to cooperating peers and stored. However, snapshot-based systems can be implemented such that they provide storage and bandwidth efficiency comparable to that of incremental backup systems, while still allowing for constant-time restoration. Namely, *single-instance storage* is a technique that has been used to provide these benefits to a number of backup [8, 24, 36], archival [33, 40] and revision control systems [26], as well as distributed file systems [29, 2].

Single-instance storage consists in storing only once any given data unit. Thus, it can be thought of as a form of compression among several data units. In a file system, where the "data unit" is the file, this means that any given content, even when available under several file names, will be stored only once. The single-instance property may also be provided at a finer-grain level, thus allowing for improved compression.

The authors of Pastiche and PeerStore argue that single-instance storage can even be beneficial at the scale of the aggregated store made of each contributor store. In essence, they assume that a lot of data is common to several participants, and thus argue that enforcing single-instance of this data at a global scale can significantly improve storage efficiency.

While common data may easily be found among participants in the context of Pastiche and PeerStore, where each participant is expected to back up their whole disk (i.e., including application binaries and data), this is certainly not the general case. For example, the mobile users of MoSAIC are expected to explicitly pay attention to their personal, critical data which are unlikely to be shared among several participants. Consequently, single-instance storage may be beneficial to mobile users only when used at a local scale, i.e., on each data owner's local store.

5 Dependability Techniques

We study, in this section, the various techniques found in the literature to address the dependability issues presented in section 2.2.

[2]There may be exceptions to this in some application scenarios where mobility patterns are known in advance. For instance, when two mobile device users take the same train every single morning while commuting.

5.1 Integrity and Consistency

Integrity and consistency are two properties that are usually obtained using some kind of data encoding. Apart from CBS, every system studied here systematically fragments the backup files. This is necessary for load-balancing: with small sized fragments, it is easier to adapt the placement to fairly balance the load imposed on the participants.

pStore uses simple data structures to encode the backup files. The files are fragmented in varying size blocks. Along with the blocks themselves, each node also stores a list of blocks that contains, for each version of the considered file, the list of the identifiers of its constituent blocks. Each block list is indexed with a structure containing a cryptographic hash of the file path and the key of its owner. There is thus one namespace per user. In practice, for the restoration of a given file, one needs to know the file path and the key of the owner. Without this metainformation, one cannot access the file's block list and consequently its blocks. The same technique is used in PeerStore, and a similar technique in Pastiche.

In ABS, each fragment is stored along with a block of meta-information about the file from which the block originates, as well as the position of the fragment within the file. These data (fragment and metainformation) are indexed using a cryptographic hash of the fragment to implement single instance storage of each fragment. The metainformation is encrypted using the owner's public key and the set (fragment and metainformation) with the owner's private key. These signatures are used to certify ownership and for ensuring integrity of the blocks.

In a similar manner, Venti-DHash encoding is based on Venti. As with a classical file system, the files are represented as trees whose leaves are the file fragments. Here, all the blocks are indexed using their digest. They are fixed-sized and the underlying storage middleware is not aware of their semantics (leaf nodes, intermediary nodes, data, metadata, etc.). To be able to restore a file, only the knowledge of the identifier of the root node is necessary.

All these techniques provide some guarantee of data block integrity since block addressing is realized using an identifier that depends on the content of the block (using a digest). When a block is restored, one can then check whether or not it is the requested block and if it is correct. If the metainformation concerning a file is stored using the same technique, integrity is thus also guaranteed file-wise. However, from the user point of view, several files can have common semantics and thus should form a consistency unit. Only Pastiche guarantees inter-file consistency. Since it is implemented as a file system, Pastiche can create shadow copies of the blocks being backed-up, so that they can be modified during the backup process without compromising their consistency.

5.2 Confidentiality and Privacy

Most of the systems studied here, like many file sharing systems, use *convergent encryption* to provide some confidentiality despite untrustworthy partner nodes. The objective is to have a ciphering mechanism that does not depend on the node performing the encryption, i.e., that is compatible with single-instance-storage. Convergent encryption is a symmetric encryption technique whose key is a digest of the block to be ciphered. The ciphered block can then be stored on the untrustworthy partner nodes. A digest of the ciphered block is commonly used as an identifier of that block. The tuple of digests (ciphered/unciphered block digests) is called digest-key or CHK for "content hash key". The data owner needs the CHK to be able to locate and uncipher a block. The CHKs are themselves backed up and ultimately the data owner only has to "remember" one CHK. Generally this ultimate CHK is stored using a secret that the data owner cannot forget, like his ID for instance. It is important to note that this technique can lead to some loss in privacy for the data owner. Indeed, when several nodes own the same block, since the ciphering scheme depends on the content and not on the nodes, they produce the same ciphered blocks, and the same CHKs. Thus they are each able to know that they share a file.

It is important to note that when single-instance-storage is not considered, it is much simpler to use classic encryption techniques, e.g., based on asymmetric ciphering.

5.3 Data Availability

In this section, we explore the techniques described in the literature for improving data availability despite failures while optimizing the use of the system resources: data replication and garbage collection.

5.3.1 Data Replication

For the systems that distribute the data among a specific set of participants (Pastiche, PeerStore, CBS, Flashback), the replication mechanism is quite simple. In Pastiche, each participant entering the system looks for 5 other participants having a lot of data in common with itself. These 5 participants then become its backup partners. It can thus tolerate 4 node failures. With PeerStore, the choice of partners is done in a different manner. However, the authors say that there are ideally as many partners as there are data replicas, which is similar to Pastiche. For the systems based on DHTs, thanks to (or because of) the properties of DHTs, the global set of data stored is homogeneously distributed among the nodes. Consequently, to tolerate the departure or the failure of a participating node, the data *has to* be replicated. In practice, the data blocks are generally replicated by the node that is responsible for it (with the closest ID) on a small number of its neighbors in the identifier space. Additionally, the block can also be kept in cache on the nodes that are on the path between the owner and the node responsible for it. ABS, among the systems based on DHTs, proposes an alternative. The data owner can choose the key under which a block will be stored. When a new block is inserted in the DHT, an attempt is made to insert it with a digest of the data as the key. A digest of the key itself (this is called rehashing) can also be used to store the block on some other participant in order to move the data or to tolerate a departure or crash.

Coding techniques are also used to finely control the level of data redundancy. Many different error-correcting coding techniques can be used: erasure codes [39] like Tornado [3], Fountain codes (also called rateless erasure codes) [4] like Raptor [37], etc. The idea of erasure codes is basically that each data block is fragmented into k fragments. From these k fragments, r other redundancy fragments are computed. From these $k + r$ fragments, any k fragments are sufficient to rebuild the original data block.

Blocks are thus used to produce fragments with a controlled level of redundancy. Venti-DHash uses this technique and stores the fragments on the successors of the node responsible for the original data block. MoSAIC also uses erasure codes for the production of redundant fragments but distributes them opportunistically to the nodes encountered.

5.3.2 Garbage Collection

Pastiche, pStore and ABS offer the possibility to delete the backed up data. Only the data owner can request this operation - requests must be signed with the owner's private key. Additionally, when single instance storage is used, as a block can be stored for several owners, an owner list is associated to each data block to permit its deletion only when every owner has requested it. In PeerStore, however, such an owner list does not exist (or is incomplete), i.e., other nodes can rely on a block for their own backup without having notified the node actually storing this block. This is due to the way PeerStore implements inter-node single-instance-storage. For this reason PeerStore does not allow delete operations. FlashBack uses the notion of a "lease" whereby a data block is stored for a given duration. This duration is determined a priori and exceeds the expected duration of unavailability of the data owner. Leases can be renegotiated when they are half-expired.

5.4 Service Availability

Failures of a cooperative backup system can lead to the loss of some of the stored data, as discussed in the previous section, but can also lead to the unavailability of the entire backup system, which we address in this section. Resilience to malicious denial-of-service attacks is a wide and active research field. The approaches used to mitigate the lack of trust between the participating nodes and to tolerate these DoS attacks can be based either on the notion of reputation (a level of the trust of the partners that can be acquired either locally or transitively) or on the use of micro-economy (exchange of checks, tokens, etc.) [14, 23, 32].

We concentrate here on the attacks that are specific to cooperative backup in general and more specifically the ones we found in the cooperative backup system we studied: selfishness and retention of backup data.

5.4.1 Selfishness

Selfishness is a problem for every resource sharing system, as we saw in section 2.2.4. Some mechanism is required to enforce fairness amongst peers - that they contribute in proportion to what they consume. Many different solutions have been proposed, most of them being based on the notion of micro-economy. We look here only at the solutions adapted to storage systems.

It is worth noting that it is not possible for a system based on DHTs to guarantee that the participants fairly contribute to the system with respect to the amount of resources they consume (see section 4.1). Consequently, Venti-DHash and pStore are not resilient to this type of attack. The ABS rehashing technique (see section 5.3.1) can be used to balance the loads on the DHT but it does not take the effective usage of each node into account.

PeerStore proposes a simple solution based on pair-wise symmetrical exchanges, i.e., each one of the two partners offers (approximately) the same storage capacity that it uses. To find partners, newcomers broadcast an offer for a given storage capacity and listen to other participant replies that offer some capacity in exchange that may be different. It is then up to newcomers to decide whether or not to accept an offer. CBS also imposes symmetrical exchange relationships, restricting data placement.

Pastiche does not deal with this problem but Samsara does: it extends the notion of symmetrical exchanges with the use of *claims*. The data owner issues a claim for the node that accepts to store its data, this exchange constitutes a contract. The value of the claim represents the storage capacity of the stored data. These claims can be forwarded to another contributor when the contributor needs to store some of its own data. Finally, each node periodically checks its co-contractors to ensure that they are adhering to the contract, i.e., to verify that its claims are satisfied, by challenging its contributors. If a node breaches a contract, its partner is free to drop its data. The use of challenges can be seen as a way to compute locally a level of reputation for a contributor.

Another simple solution is proposed in CFS [10]: each contributor limits any individual peer to a fixed fraction of its space. These quotas are independent of the peer's space contribution. CFS uses IP addresses to identify peers, and requires peers to respond to a nonce message to confirm a request for storage, preventing simple forgery. This does not defend against malicious parties who have the authority to assign multiple IP addresses within a block, and may fail in the presence of network renumbering.

Several of these solutions were proposed to be extended with trusted third parties, either centralized or distributed among trusted hardware devices. For instance, PAST [35] provides quota enforcement that relies on a smartcard at each peer. The smartcard is issued by a trusted third party, and contains a certified copy of the node's public key along with a storage quota. The quota could be determined based on the amount of storage the peer contributes, as verified by the certification authority. The smartcard is involved in each storage and reclamation event, and so can track the peer's usage over time. Fileteller [19] proposes the use of micro-payments to account for storage contributed and consumed. Such micro-payments can provide the proper incentives for good behavior, but must be provisioned and cleared by a third party and require secure identities.

It is worth noting that, as a side effect, solutions based on symmetrical exchanges have the advantage of being resilient to flooding attacks, whereby a node tries to obtain many storage resources by flooding the network with requests. On the contrary, DHT based systems are not resilient to this type of attack due to the very nature of DHTs.

5.4.2 Retention of Backup Data

Data retention is the situation in which a contributor does not release backed up data when an owner issues a restoration request. This can be non intentional, e.g., the contributor has crashed, or is disconnected, or intentional/malicious, e.g., the contributor did not actually store the data or tries to blackmail the data owner. Generally speaking, unintentional retention should be tolerated whereas malicious retention should be prevented, or even punished.

In CBS, there is a two-fold solution to these problems: first there are periodic challenges to verify that the partners really do store the data for which they are responsible for, and second, there are rules to tolerate temporary node failures. The periodic challenges are actually read requests for randomly chosen data blocks sent to the contributors by data owners. Tolerance of temporary faults is based on a *grace period* during which a participant can be legitimately unavailable. After expiration of the grace period, the data stored for the disconnected node can be erased (the data owner locally decides to associate a bad reputation to the contributor). However, the grace period can be used to gain resources dishonestly without contributing to the system. A countermeasure is to define a *trial period*, that is longer than the grace period, during which backup and challenges are permitted but restoration is not.

This challenge technique is also used by the other studied systems, in an optimized form: a challenge concerns several blocks at a time and the response is a signature of the set of blocks [9] [24].

Samsara and PeerStore also have a slightly different way of punishing *unavailable* nodes: their blocks are progressively deleted. The probability of deletion of a block is chosen such that, given the number of block replicas, the probability of all the replicas being deleted becomes significant only after a large number of unsatisfied challenges.

6 Conclusion

Peer-to-peer/cooperative systems constitute a new emerging approach for the design of heavily distributed systems. They have very good properties regarding scalability and are thus particularly well-adapted to ubiquitous computing scenarios. The application of peer-to-peer coopearation to backup has been rendered feasible by recent dramatic increases in storage capacity and network bandwidth. In this paper, we have surveyed the technical solutions to this problem.

We first observed that the field of cooperative backup for wide-area networks or local-area networks is very active. This research field has been recently boosted by the peer-to-peer trend and reused many of the P2P techniques: distributed hash tables, single-instance-storage, convergent encryption, etc. However, very little work has targeted mobile devices, even if cooperative backup seems to be quite appropriate for them (new data is frequently produced on many types of devices, even disconnected from the fixed infrastructure: digital cameras, phones, PDAs, laptops. However, mobile devices have their specificities (ephemeral connections, reduced energy, etc.), so many of the techniques developed for WANs and LANs cannot be applied. Much effort is still needed to alleviate the specific problems raised by frequent disconnections, ephemeral connections, limited battery power, inability to access trusted third parties, etc.

From this situation, trails that can be followed to make some progress in this field include: adequate disconnected cooperation incitatives, proper erasure codes with varying parameters, realistic mobility models, and stochastic models of the dependability of mobile devices implementing cooperative services.

References

[1] C. Batten, K. Barr, A. Saraf, and S. Treptin. pStore: a secure peer-to-peer backup system. Technical Report MIT-LCS-TM-632, MIT Laboratory for Computer Science, December 2001.

[2] W. J. Bolosky, J. R. Douceur, D. Ely, and M. Theimer. Feasibility of a serverless distributed file system deployed on an existing set of desktop pcs. In *Proceedings of the International Conference on Measurement and Modeling of Computer Systems*, pages 34–43, 2000.

[3] J. W. Byers, M. Luby, and M. Mitzenmacher. Accessing multiple mirror sites in parallel: Using Tornado codes to speed up downloads. In *INFOCOM (1)*, pages 275–283, 1999.

[4] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *SIGCOMM*, pages 56–67, 1998.

[5] A. Chervenak, V. Vellanki, and Z. Kurmas. Protecting file systems: A survey of backup techniques. In *Proceedings Joint NASA and IEEE Mass Storage Conference*, March 1998.

[6] J. Cooley, C. Taylor, and A. Peacock. ABS: the apportioned backup system. Technical report, MIT Laboratory for Computer Science, 2004.

[7] B. F. Cooper and H. Garcia-Molina. Bidding for storage space in a peer-to-peer data preservation system. In *ICDCS*, pages 372–, 2002.

[8] L. P. Cox and B. D. Noble. Pastiche: making backup cheap and easy. In *Fifth USENIX Symposium on Operating Systems Design and Implementation*, pages 285–298, Boston, MA, USA, December 2002.

[9] L. P. Cox and B. D. Noble. Samsara: honor among thieves in peer-to-peer storage. In *Proceedings 19th ACM Symposium on Operating Systems Principles*, pages 120–132, Bolton Landing, NY, USA, October 2003.

[10] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings 18th ACM Symposium on Operating Systems Principles*, pages 202–215, October 2001.

[11] S. Elnikety, M. Lillibridge, and M. Burrows. Peer-to-peer cooperative backup system. In *The USENIX Conference on File and Storage Technologies*, Monterey, California, USA, January 2002.

[12] A. V. Goldberg and P. N. Yianilos. Towards an archival intermemory. In *Proceedings IEEE International Forum on Research and Technology Advances in Digital Libraries (ADL'98)*, pages 147–156. IEEE Society, April 1998.

[13] G. M. Greco and L. Floridi. The tragedy of the digital commons. *Ethics and Information Technology*, 6(2):73, 2003.

[14] C. Grothoff. An excess-based economic model for resource allocation in peer-to-peer networks. *Wirtschaftsinformatik*, June 2003.

[15] E. Growchowski. Emerging trends in data storage on magnetic hard disk drives. In *Datatech*, pages 11–16. ICG Publishing, September 1998.

[16] G. Hardin. The tragedy of the commons. *Science*, 162:1243–1248, 1968.

[17] O. Heckmann and A. Bock. The eDonkey 2000 Protocol. Technical Report KOM-TR-08-2002, Multimedia Communications Lab, Darmstadt University of Technology, Dec. 2002.

[18] E. Hsu, J. Mellen, and P. Naresh. DIBS: distributed backup for local area networks. Technical report, Parallel & Distributed Operating Systems Group, MIT, USA, 2004.

[19] J. Ioannidis, S. Ioannidis, A. D. Keromytis, and V. Prevelakis. Fileteller: Paying and getting paid for file storage. In *Sixth Annual Conference on Financial Cryptography*, page 282299, Bermuda, March 2002.

[20] F. Junqueira, R. Bhagwan, K. Marzullo, S. Savage, and G. M. Voelker. The Phoenix recovery system: rebuilding from the ashes of an internet catastrophe. In *Ninth Workshop on Hot Topics in Operating Systems (HotOS IX)*, 2003.

[21] M.-O. Killijian, D. Powell, M. Banâtre, P. Couderc, and Y. Roudier. Collaborative backup for dependable mobile applications. In *Proceedings of 2nd International Workshop on Middleware for Pervasive and Ad-Hoc Computing (Middleware 2004)*, pages 146–149, Toronto, Ontario, Canada, October 2004. ACM.

[22] 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 the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, pages 190–201, November 2000.

[23] K. Lai, M. Feldman, J. Chuang, and I. Stoica. Incentives for cooperation in peer-to-peer networks. In *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[24] M. Landers, H. Zhang, and K.-L. Tan. PeerStore: better performance by relaxing in peer-to-peer backup. In *Proceedings of the Fourth International Conference on Peer-to-Peer Computing*, pages 72–79, Zurich, Switzerland, August 2004.

[25] B. T. Loo, A. LaMarca, and G. Borriello. Peer-to-peer backup for personal area networks. Technical Report IRS-TR-02-015, UC Berkeley; Intel Seattle Research (USA), May 2003.

[26] T. Lord. The GNU arch distributed revision control system, 2005.

[27] K. Lua, J. Crowcroft, M. Pias, R. Sharma, and S. Lim. A survey and comparison of peer-to-peer overlay network schemes. *Communications Surveys & Tutorials, IEEE*, pages 72–93, 2005.

[28] P. Maymounkov and D. Mazières. Kademlia: A peer-to-peer information system based on the XOR metric. *Lecture Notes in Computer Science*, 2429:53–??, 2002.

[29] A. Muthitacharoen, B. Chen, and D. Mazières. A low-bandwidth network file system. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles*, pages 174–187, October 2001.

[30] A. Muthitacharoen, R. Morris, T. M. Gil, and B. Chen. Ivy: a read/write peer-to-peer file system. *SIGOPS Oper. Syst. Rev.*, 36(SI):31–44, 2002.

[31] Napster. Site internet napster : http://www.napster.com.

[32] N. Oualha and Y. Roudier. Cooperation incentive schemes and validation techniques. Technical report, Institut Eurecom, June 2006.

[33] S. Quinlan and S. Dorward. Venti: a new approach to archival storage. In *Proceedings of the First USENIX Conference on File and Storage Technologies*, pages 89–101, Monterey,CA, 2002.

[34] C. Randriamaro, O. Soyez, G. Utard, and F. Wlazinski. Data distribution in a peer to peer storage system. *Journal of Grid Computing (JoGC), Special issue on Global and Peer-to-Peer Computing, Springer,Lecture Notes in Computer Science*, 2006.

[35] A. Rowstron and P. Druschel. Storage management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. October 2001.

[36] M. Rubel. Rsnapshot: a remote filesystem snapshot utility based on rsync, 2005.

[37] A. Shokrollahi. Raptor codes, 2003.

[38] E. Sit, J. Cates, and R. Cox. A DHT-based backup system. Technical report, MIT Laboratory for Computer Science, August 2003.

[39] H. Weatherspoon and J. Kubiatowicz. Erasure coding vs. replication: A quantitative comparison. In *Proceedings of the First International Workshop on Peer-to-Peer Systems*, March 2002.

[40] L. L. You, K. T. Pollack, , and D. D. E. Long. Deep Store: an archival storage system architecture. In *Proceedings of the 21st International Conference on Data Engineering (ICDE 2005)*, pages 804–815, Tokyo, Japan, April 2005. IEEE.

A Survey of Cooperation Incentive Schemes

Nouha OUALHA, Yves ROUDIER

Institut Eurécom

Contents

Introduction

Decentralized system algorithms and protocols have recently received a lot of interest in mobile ad-hoc networks as well as in peer-to-peer (P2P) systems. The development of such techniques is a necessity to be able to attain cost-effective and reliable applications in this setting, yet it brings up far-reaching issues that have to be dealt with. In decentralized systems, decision-making may not be located at a specific and central group of devices (repeaters, bridges, routers, gateways, servers) but can be distributed to end-user devices. Decisions and actions may use the computing power, bandwidth, and disk storage space of all the participants (peers) in the network rather than being concentrated in a relatively low number of special devices. The decentralized structure makes it possible to achieve minimal administrative and operational costs. Peers in this type of system normally have equivalent responsibilities and privileges. The intricate notions of self-organization and self-management require that each peer provide his own contribution for the correct operation of the system.

The idea of handing basic mechanisms of the system over to autonomous peers raises new concerns, in particular with respect to the establishment of trust between peers, to the stimulation of their cooperation, and to the fairness of their respective contributions. Self-organization opens up new security breaches because a peer must be able to defend against others perpetrating new forms of denial of service. Selfishness, as illustrated by the so-called free-riding attack, is a first type of such threats in which the attacker (called free-rider) benefits from the system without contributing its fair share. Systems vulnerable to free-riding either run at reduced capacity or collapse entirely because the costs of the system weigh more and more heavily on the remaining honest peers encouraging them to either quit or free ride themselves. Flooding is a second type of denial of service: the attack can be launched by sending a large number of query messages asking for resources to a victim peer in order to slow it until it is unusable or crashes. For example, an attacker can attempt to make a lot of read and write operations in a distributed storage application. Cheating (or retention) is a third form of denial of service in which the attacker retains data required for the system to work or does not comply with the normal course of action in order to obtain an unfair advantage over other peers. So-called "cooperation enforcement" mechanisms (which should more properly be called cooperation incentive schemes) provide ways of managing and organizing resources and aim at dealing with the security challenges that traditional security approaches (e.g. authentication, access control) can not cope with.

The following sections introduce motivating applications for cooperation incentives, then detail how incentive schemes work, and finally discuss how these schemes may be validated.

Applications

Cooperation incentives mechanisms are present in various application domains. It is generally suggested that cooperation will help entities to succeed better than via competition. In [16], Buttyán demonstrated that the best performance in routing is obtained when nodes are very cooperative. In a cooperation incentive mechanism, cooperative behavior should be more beneficial than an uncooperative behavior. The two main categories of incentives are reputation and remuneration. This section describes several applications that benefit from cooperation enforcement.

1. Ad-hoc packet routing

Multi-hop ad-hoc networks, as frequently referred to under the term MANETs (Mobile Ad hoc NETworks), can be set up rapidly and spontaneously. Connections are possible over multiple nodes. These nodes operate in a decentralized and self-organizing manner and do not rely on a fixed network topology. Intermediate nodes in a route have to act as routers to forward traffic towards its destination. To achieve this operation, incentives for cooperation between nodes become a requirement, because rational users would rather preserve the energy of their personal devices rather than spend it on cooperative routing. There has been a wealth of work on cooperative network forwarding. In the Watchdog/Pathrater [38] scheme, the watchdog detects non-forwarding nodes by overhearing the transmission, and the pathrater keeps a rating of every node and updates it regularly. The two components enable nodes to route messages avoiding misbehaving nodes in their route. Misbehaving nodes are detected and avoided in the routing path but not punished. In CONFIDANT (Cooperation Of Nodes, Fairness in Dynamic Ad-hoc NeTworks) [35], the response to misbehaving nodes is more severe than just avoiding them for routing; it also denies them cooperation. Similarly to Watchdog/Pathrater, in CONFIDANT reputation is self-carried by nodes. Nodes monitor their neighborhood and gather second-hand information from others. By Bayesian estimation, they classify others as normal or misbehaving. In CORE (COllaborative REputation) [32], the information collected is classified into subjective reputation (direct information), indirect reputation (positive reports from other nodes), and functional reputation (task-specific information). The combined reputation value is used to make decisions regarding a given node, that is to either cooperate with it or gradually isolate it. TermiNodes ([15] and [16]) uses a different approach based on a tamper-proof security module for each node maintaining a nuglet counter. When the node wants to send a packet, it decreases its nuglet value by a number of credits proportional to the estimated number of intermediate nodes in the route. When the node forwards a packet, its nuglet purse becomes bigger. While TermiNodes uses a tamper-proof hardware placed at each node, Sprite [39] does not require any tamper-proof hardware at any node. Sprite is based on a central Credit Clearance Service (CCS). Every node is supposed to have a digital ID obtained from a Certification Authority (CA). When a node receives a message, the node keeps a receipt of the message. Sprite assumes that every source node knows the entire path to the destination node through a secure ad hoc routing protocol based upon DSR. The underlying ad hoc routing protocol only exists for packet delivery, not for routing decision making. When the node has a fast connection to the CCS, which is reachable via an overlay network, it reports to the CCS the messages that it has received/forwarded by uploading its receipts. Depending on the reported receipts of a message, the CCS then determines the charge and credit to each node involved in the transmission of a message. [39] introduces a formal model in order to prove the effectiveness of Sprite in restraining selfish behavior at the network layer, however Sprite presents a weakness for it relies on the accessibility of the CCS.

2. P2P applications

a) File sharing

Peer-to-peer file sharing has become widespread over the Internet, accounting for almost 80% of total traffic [26]. The flagship of P2P applications is file sharing. In the beginning, P2P applications used to provide content sharing services. The early networks providing this service were based on Napster[1] protocol. These networks were hybrid peer-to-peer networks in which the index service is provided centrally by a coordinating entity, the Napster server. The functionality of the server is to deliver to a requesting peer a peers' list having the desired requested MP3 files. Then, the peer can obtain the respective files directly from the peer offering them. In contrast, Gnutella[2] functions without any central coordination authority. Search requests are flooded into the network until the TTL (Time-To-Live hop counter) of the message has expired or the requested file has been located. Positive search results are sent to the requesting peer who can then download the file directly from the peer offering it. Both Napster and Gnutella focus more on information retrieval than on publishing. Freenet [11] provides anonymous publication and retrieval of data. Anonymity is provided through several means encrypted search keys and source-node spoofing. In Freenet, when peer storage is exhausted, files are deleted according to the least-recently-used principal so the system keeps only the most requested documents. Another drawback is the complexity of file search

[1] http://www.napster.com/

[2] http://www.gnutella.com/

process. In fact there is a significant difference between Freenet and the systems presented so far which is that files are not stored on the hard disk of the peers providing them, but they are intentionally stored at other locations in the network. They are stored at peers having the numerically closest identification number to their IDs. The document lookup is a routing model based on keys to locate data similarly to Distributed Hash Tables (DHT). Free riding has been notably observed in such applications, and first attempts at using reputation incentives to counter it were made in systems like NICE [37]. NICE is a platform for implementing cooperative distributed applications, in particular P2P applications. The NICE system aims at identifying the existence of cooperative peers; it claims to efficiently locate the minority of cooperating users, and to form a clique of users all of whom offer local services to the community. The system is based on peer reputation which is stored in the form of cookies. Peer cookies express if a peer correctly handled its transactions (transactions consist of secure exchanges of resource certificates) with other peers. Before initiating a transaction, every peer checks locally the cookie that's speaks about the target peer and that indicates if the latter can be trusted. However, if no cookie is available for that peer, cooperation with other peers in acquiring that information is necessary.

b) P2P file system

A generation of P2P applications uses the promising DHT-based overlay networks. DHTs such as CAN, Chord, Pastry, and Tapestry allow a decentralized, scalable, and failure-tolerant storage. Well-known approaches are PAST [30] based on Pastry and OceanStore [12] based on Tapestry. Each PAST node can act as a storage node and a client access point. These schemes have basic similarities in the way they are constructed. Participants receive a public/private key pair. Keys are used to create an unambiguous identification number for each peer and for the stored files with the aid of a hash function. To take profit of the storage, a peer has to pay a fee or to make available its own storage space. Keys generation and distribution and monitoring are handled by “special” peers who have to be highly capable and highly available. A further correspondence within these DHT-based schemes is that the storage system has as primary function data backup. The service is provided by means of file replication and random distribution of identification numbers to peers. The procedure guarantees geographically-separated replicas which increases the availability of a given file. Compared with PAST [30] and OceanStore [12], Free Haven [34] is designed for more anonymity and persistence of documents than for frequent querying. An author in Free Haven generates a public/private keys pair, signs his document fragments, and uploads them into the server. Each server hosts data from the other servers in exchange for the opportunity to store data of its own into the community of servers, servnet. Trading of document fragments adds to author anonymity. When a reader wishes to retrieve a document from the servnet, he requests it from any server, including a location and key which can be used to deliver the document in a private manner. This server broadcasts the request to all other servers, and those which are holding shares for that document encrypt them and deliver them to the reader's location.

c) P2P backup

The latest generation of peer-to-peer systems is a generation of storage systems having data backup as its primary function. Pastiche [17] is based on Pastry for locating nodes and exploits excess disk capacity to perform peer-to-peer backup with no administrative costs. Each Pastiche node minimizes storage overhead by selecting peers that share a significant amount of data. It replicates its archival data on more than one peer. Most of these replicas are placed nearby to ease network overhead and minimize restoration time. To address the problem of storing data on malicious nodes, Pastiche uses a probabilistic mechanism to detect missing backup state by periodically querying peers for stored data. However it sacrifices a fair amount of privacy because one node can grab some information about the backup data. This issue is less critical for the CIBS (Cooperative Internet Backup Scheme) [20] scheme where fragments of a file are stored at different geographical locations, and partners are tracked by a central server. To ensure a high reliability, the scheme adds redundancy through Reed-Solomon erasure correcting code.

3. Wireless applications

a) Wireless P2P backup

Another P2P backup scheme but this time a wireless system is the Flashback [5] application which has been proposed for Personal Area Network (PAN). Flashback is a solution designed to provide peer-to-peer power-aware backup for self-managing, mobile, impoverished devices. In Flashback each device has an identifier assigned out-of-band during the installation of the Flashback. To ensure that devices only participate to the PANs to which they are assigned, a lightweight certificate-based authentication scheme is used. In order to keep track of replicas, Flashback maintains in persistent storage a replica table. It uses an opportunistic model to replicate local data given the constraints imposed by the power and storage resources.

b) Wireless information and access services

Wired peer-to-peer storage applications are probably the most widespread peer-to-peer applications. For wireless networks, the most popular applications are information services. There are three approaches to provide information services for wireless devices. The first approach consists of the wireless Internet access offered by the new wireless technologies like 3G or 802.11. The second approach which is information servers was first proposed by the DataMan project [8]. Information services such as e-mail, fax and web access are supplied by placing these servers called info-stations at traffic lights, building entrances and airport lounges. The third approach is based on peer-to-peer data dissemination among mobile users. 7DS [21] is an application based on the last approach. It allows a peer to browse the content of the cache of a peer that has been made accessible to it to search for URLs or keywords. This operation can be performed in two modes: on-demand mode or prefetching mode. In the prefetch mode, 7DS anticipates the information needs of the peer. In the on-demand mode, it searches for the information or the URL when the peer requests it. Mobile devices in 7DS do not need any base stations to gain access to the service.

4. Commercial web sites

a) Auction sites

Auction sites allow sellers to list items for sale, buyers to bid for these items, then the items to be sold to the highest bidder. In general, the person who puts the item up for auction pays a fee to the auctioneer. In some cases, there is a minimum or reserve price; if the bidding does not reach the minimum, the item is not sold. Reputation systems are used in auctioning in order to help users making good choices when selecting transacting partners. EBay[3] is one popular online auction site. The feedback forum on eBay allows sellers and buyers to rate each other as positive, negative, or neutral. Ratings of buyers and sellers are conducted after the completion of a transaction, which is monitored by eBay. The reputation system relies on a centralized repository that stores and manages ratings. The overall reputation of a participant is the sum of ratings about him over the last 6 months. EBay also provides one month old and seven day old ratings to let users know about recent behavior of the participant. The EBay system makes it possible to perform fake transactions even though at a cost for eBay charges a fee for listing items: still, this opens up opportunities to acquire undue ratings.

[3] http://ebay.com/

b) Review and recommendation sites

In review sites, individual reviewers, who are generally individuals, are providing information to fellow consumers. In these systems, a reputation rating is applied to both products and reviewers themselves, in particular to discourage product bashing. One example of such a system is Amazon[4], an online bookstore that allows members to write book reviews. A user can become an Amazon member by simply signing up. Reviewers reviews of a book are made of some text and a rating in the range of 1 to 5 stars. Members and users rate reviews as being helpful or not. Amazon ranks reviewers based on their rating and other parameters (which are not publicly revealed). Reviewers with a high ranking are given the status of top reviewers. To reduce repetitive ratings from the same users, Amazon only allows one vote per registered cookie for any given review. Another review site is Epinions[5]. This web site charges product manufactures and online shops by the number of clicks consumers generate as a result of reading about their products on Epinions's web site. Amazon does not give any financial incentive for well-reputed reviewers. Top reviewers however are paid in Epinions. Except for this financial plus to reviewers, the reputation system of Epinions operates in the same way as in Amazon with some little non-important differences.

c) Web page ranking system

Early web search engines simply presented all web pages that matched the keywords entered by the user without ranking them, which often results in too many and irrelevant pages being listed. PageRank [18] was proposed to rank web pages based on page reputation. Google[6]'s search engine is based on this algorithm. The PageRank of a web page *u* is given in [18] as:

$$R(u) = cE(u) + c \sum_{v \in N^-(u)} \frac{R(v)}{|N^+(v)|}$$

where $N^-(u)$ denotes the set of web pages pointing to *u*, $N^+(v)$ denotes the set of web pages that *v* points to, and *E* corresponds to a source of rank. So, a hyperlink is a positive referral of the page it points to. Negative referrals do not exist because it is impossible to blacklist a web page using the above equation. The public PageRank measure[7] does not fully describe Google's page ranking algorithm, which takes into account other parameters for the purpose of making it difficult or expensive to deliberately influence ranking results in what can be seen as a form of "spamming" (this term has prevailed to denote such a behavior in a similar way to email spam that pollutes one's email box).

Principles

Cooperation is a central feature of decentralized systems, and even more so ad-hoc ones, to compensate for the lack of a central and dedicated entity and still achieve some general function. However, cooperation to achieve some functionality may be hampered by the fact that users have full authority on their devices and, as proven by experience, will on average try to maximize the benefits they get from the network. In general, the cooperative behavior of a device will indeed result in an increase in its resource consumption or missed opportunities to take more than its fair share of a resource (e.g. network, CPU, storage space). In case of mobile ad hoc forwarding for instance, the node forwarder is confronted with additional energy and bandwidth usage for reception and transmission of packets, as well as with the increase of computational resource consumption. Knowing that mobile devices have inherently scarce resources, each of these devices has better not cooperate from its point of view. In case of file sharing applications, a node can take advantage over the system by downloading files without contributing to it. To counterbalance this, and achieve an overall

[4] http://www.amazon.com/
[5] http://www.epinions.com/
[6] http://www.google.com/
[7] http://pr.blogflux.com/

better result, it is primordial to design incentive mechanisms for cooperation that discourage uncooperative behavior be it passive or malicious. At the same time, these mechanisms can not prevent the non cooperative behavior of devices due to valid and reasonable reasons (e.g. crashing, energy shortage, route breaks), which should normally not be punished similarly to malicious non-cooperation.

As seen in section B, there are many cooperation incentive schemes which are diverse not only in terms of the applications for which they are employed, but also in terms of the features they implement, the type of reward and punishment used, and their operation over time. Obreiter et al. [33] classified cooperation enforcement mechanisms by essentially differentiating trust-based patterns from trade-based patterns. The authors make a distinction between static trust which is about pre-established trustworthiness between peers, and dynamic trust which refers to reputation-based trust. In trade-based patterns, remuneration is the central notion and can be immediate, which they term barter trade, or deferred, which they term bond-based. While this classification was the first to relate many incentive mechanisms together, we do not agree with some naming choices, nor with the relationship of trust and incentives put forward by the authors. Trust reflects the individual view of an entity about another entity's trustworthiness. Whatever the incentive mechanism, an entity will have to ask itself whether it trusts another entity to cooperate with it. Trust establishment easily maps to reputation systems but may use remuneration systems as well. We similarly classify cooperation mechanisms into two types: remuneration-based and reputation-based mechanisms (corresponding respectively to trust-based and trade-based patterns in the Obreiter et al. paper [33]). We describe them one by one in the following.

5. Reputation based mechanisms

In reputation-based mechanisms, the decision to interact with a peer is based on its reputation. Reputation mechanisms need reputation management systems for which the architecture is either centralized, or decentralized, or both.

a) Reputation based system architecture

The estimation of reputation can be performed either centrally or in a distributed fashion. In a centralized reputation system, the central authority that collects information about peers typically derives a reputation score for every participant and makes all scores available online. In a distributed reputation system, there is no a central authority for submitting ratings or obtaining reputation scores of others. However it might be some kind of distributed stores where ratings can be submitted. One example of such architecture is FastTrack [13] architecture which is used in P2P networks like KaZaA[8], Grokster[9], and iMesh[10]. These networks have two-tier hierarchy consisting of ordinary nodes (ONs) in the lower tier and supernodes (SNs) in the upper tier. SNs are generally more powerful in terms of connectivity, bandwidth, processing and non-NATed (Network Address Translations) accessibility. SNs keep tracks of ONs and other SNs and act as directory servers during the search phase. Such architecture can be convenient to manage peers' reputation using supernodes as distributed stores; unfortunately this is not the case in the existing FastTrack-based P2P networks. In KaZaA for example, each node has a participation level based some QoS (Quality of Service) parameters which is stored locally. The participation level score is used in prioritizing peers during periods of high demand. Most of the time in a distributed architecture, ratings are estimated autonomously by each peer. Each peer records ratings about its experiences with other peers and/or tries to obtain ratings from other parties who have had experiences with a given target peer. A good example of decentralized reputation-based approach to trust management is NICE [37]. This system searches the network on the runtime and builds a trust graph where each edge represents how much the source trusts the destination. A reputation

[8] http://www.kazaa.com/
[9] http://www.grokster.com/
[10] http://imesh.com

value is calculated based on this trust graph. Then, NICE algorithm selects a trust path based on whether it is the strongest path or using a weighted sum of strongest disjoint paths.

The centralized approach to reputation management is not fault-tolerant. In the decentralized approach, it is often impossible or too costly to obtain cooperation evaluations resulting from all interactions with a given peer. Instead reputation is based on a subset of such evaluations, usually obtained from the neighborhood. The reputation mechanism should therefore be designed such as to avoid inconsistencies.

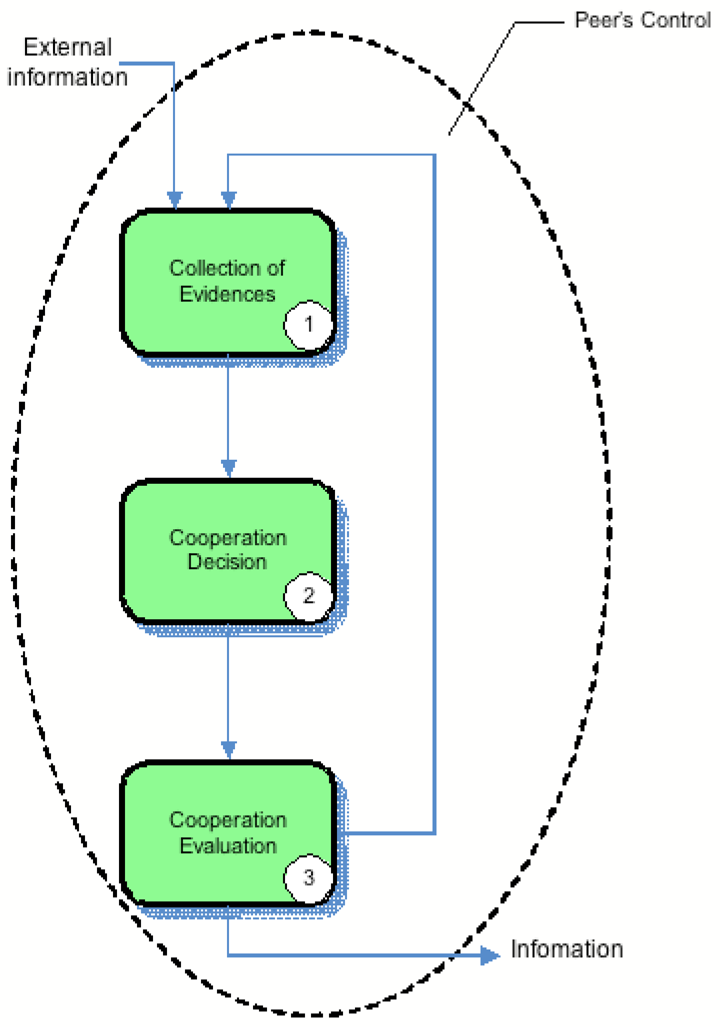

Figure 1 Reputation-based Mechanism

b) Reputation system operations

A reputation-based mechanism is composed of three phases (Figure 1):

1. **Collection of evidence:** Peer reputation is constructed based on the observation of the peer, experience with it, and/or recommendations from third parties. The semantics of the information collected can be described in terms of a specificity-generality dimension and a subjectivity-objectivity dimension.

    - Specific vs. general information: specific information about a given peer relates to the evaluation of a specific functionality or aspect of this peer such as its ability to deliver a service on time. Whereas, general information refers to all functionalities (e.g. measured as average).

- Objective vs. subjective information: A peer obtains objective information (a.k.a. direct or private information) about a given peer through his personal interactions with the considered peer and subjective information (a.k.a. indirect or public information) by listening to messages or negotiations that are intended to other peers, or by asking neighboring peers.

2. **Cooperation decision:** Based on the collected information, a peer can make a decision whether he should cooperate with a peer. He will make his decision taking account of the peer's reputation. There are various methods for computing peers' reputation. Here we describe some of them.

    - Summation or average of ratings: the simplest method to compute reputation is to compute the sum of positive ratings minus the sum of negative ratings. This is the principle used in eBay[11]'s reputation forum. Amazon uses another simple scheme where the reputation score is computed as the average of all ratings.
    - Bayesian based computation: Reputation is computed based on the previous estimated reputation with the new evaluation score. The reputation system uses the beta PDF (Probability Density Function) denoted by *beta(p |α,β)* using the gamma function *Γ*.

    $$beta(p \mid \alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)+\Gamma(\beta)} p^{\alpha-1}(1-p)^{\beta-1}; 0 \le p \le 1, \alpha > 0, \beta > 0$$

    Where α and β represent the amount of positive and negative ratings respectively, and p represents the probability variable. The PDF expresses the uncertain probability that future interactions will be positive (cooperation).
    - Flow model based computation: "systems that compute trust or reputation by transitive iteration through looped or arbitrarily long chains can be called flow models" [2]. One example is Google's PageRank where the PageRank of a web page increases if there are many hyperlinks coming to this page web, and decreases if there are many hyperlinks leaving the webpage. Another flow model is EigenTrust [36]. The EigenTrust system computes a global trust value for a peer by multiplying iteratively normalized local matrices of trust scores of each peer in the system. For a large number of matrices, all will converge to stable trust values.
    - Further reputation computation models are given in [2] sec.8. In the discrete trust model, the trustworthiness of the neighbor is taken into account before considering subjective information. The belief model relates to the probability theory. In this model, the sum of probabilities over all possible outcomes does not necessarily add up to 1, the remaining probability being interpreted as uncertainty. In fuzzy models finally, reputation and trust are considered as fuzzy logic concepts.

3. **Cooperation evaluation:** The occurrence of interaction with a peer is conditional on the precedent phase. After interaction, a node must provide an evaluation of the degree of cooperation of the peer involved in the interaction. Peers performing correct operations, that is, behaving cooperatively, are rewarded by increasing their local reputation accordingly. A peer with a bad reputation will be isolated from the functionality offered by the group of peers as a whole. The evaluation of the current interaction can convey extra information about other past interactions (piggybacking) that can be collected by the neighboring peers.

[11] http://ebay.com

c) Attacks and counter-measures

This type of mechanisms has to cope with several problems due to node misbehavior. Misbehavior ranges from simple selfishness or lack of cooperation to active attacks aiming at denial of service (DoS), attacks to functionality (e.g., subversion of traffic), and attacks to the reputation system (liars).

To guard against the impact of liars, the CORE mechanism [32] for instance takes into account only positive reputation from indirect information, together with reputation from direct information (does the node hear the packet forwarded by its peer?): defamation is thus avoided, yet unjustified praising is still possible. In a more restrictive manner, RPG (Reputation Participation Guarantee) [7] forbids the diffusion of reputation between peers. Only direct information is taken into account; selfishness is detected by sending probe packets.

A different approach, relying on indirect information, is taken in Watchdog/Pathrater [38]. A watchdog is in charge of identifying the misbehaving nodes, and a path rater is in charge of defining the best route avoiding these nodes. It is pretty much the same approach that is taken in CONFIDANT [35]: a neighborhood monitor has the role of identifying misbehavior, which is rated.

A trust manager sends and receives alarm messages to and from other trust managers, while a path manager maintains path ranking. As a result, nodes in the network will exclude misbehaving nodes by both avoiding them for routing and by denying them cooperation. So misbehaving nodes will not pay off but they will be isolated. Contrary to many proposals, Watchdog/Pathrater [38] evaluates cooperation but does not enforce it: non cooperative nodes in Watchdog/Pathrater will not be punished like in CORE [32] or CONFIDANT [35], their messages are still forwarded while they are not forced to forward the messages of the other nodes.

The systems presented so far focus on the network layer forwarding. In this type of application, cooperation evaluation is immediate, yet other mechanisms may require the evaluation to take place on a longer timescale. Reputation estimates then need to be preserved: this may mean that it is self-carried by the peer if reputation is based on direct information; otherwise, reputation should not depend on the proximity of the peer since nearby nodes are likely to move away over a long period of time. This is for instance the case for distributed backup applications.

In the CIBS (Cooperative Internet Backup Scheme) scheme [20], each computer has a set of geographically-separated partner computers that collectively hold its backed up data. In return, the computer backs up a part of its partner's data. To thwart free riding attacks, a computer can periodically challenge each of its partners by requesting him a block of the backed up data[12]. An attack can then be detected and the data blocks of the attacker that are

[12] Some disruption attacks, i.e. attacks aiming at disrupting, impairing or destroying a system or a particular user, can be avoided by limiting reads to mutually chosen random blocks.

stored in the attacked computer are consequently dropped. In this scheme, each peer takes note of its direct experience with a partner, and if this partner does not cooperate voluntarily or not beyond some threshold, the peer may decide to establish a backup contract with another partner.

Another example of reputation-based mechanisms for distributed storage is the Free Haven project [34]. The overall design of the project is based on a community of servers, called the servnet where each server hosts data from other servers in exchange of the opportunity to store data of its own in the servnet. The incentives for cooperation are based on a reputation mechanism. A trust module on each server maintains a database of each other server, logging past direct experience as well as what other servers have said.

6. Remuneration based mechanisms

In contrast to reputation-based mechanisms, remuneration based incentives are an explicit counterpart for cooperation and provide a more immediate penalty to misconduct. Remuneration brings up requirements regarding the fair exchange of the service for some form of payment [24]. This requirement in general translates to a more complex and costly implementation than for reputation mechanisms. In particular, remuneration based

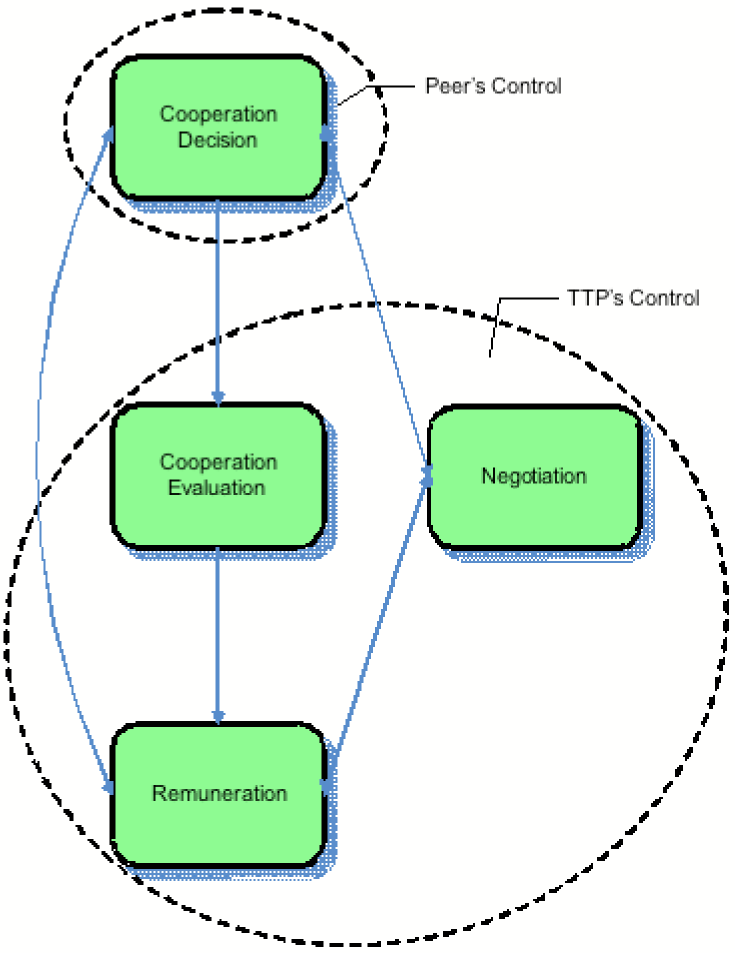

Figure 2 Remuneration-based Mechanism

mechanisms require trusted third parties (TTP) such as banks to administer remuneration of cooperative peers; these entities do not necessarily take part in the online service, but may be contacted in case of necessity to evaluate cooperation. Tamper proof hardware (TPH) like secure operating systems or smart cards have been suggested or used to enforce in a decentralized fashion the fair exchange of the remuneration against a proof that the cooperative service was undertaken by a peer node.

a) Remuneration based system architecture

A remuneration based mechanism comprises four main operations (see Figure):

- **Negotiation:** The two peers may often negotiate the terms of the interaction. Negotiating the remuneration in exchange for an enhanced service confers a substantial flexibility to the mechanism. The negotiation can be performed either between the participating peers or between peers and the authority.
- **Remuneration:** The remuneration can consist in virtual currency units (a number of points stored in a purse or counter) or real money (banking and micropayment), or bartering units (for instance quotas defining how a certain amount of resources provided by the service may be exchanged between entities). The latter can even be envisioned in the form of micropayments [23]. Regarding real money, this solution assumes that every entity possesses a bank account, and that banks are enrolled in the cooperative system, directly or indirectly through some payment scheme. The collaborating peer is remunerated by issuing a check or making a transfer of money. In the first case, remuneration implies a number of points added to a counter connected with the collaborating peer. The remuneration can be guaranteed at once or only after a certain number of steps (deposit, remuneration for data storage, remuneration for data retrieval…).
- **Cooperation decision:** The peer in a self-organizing network is always the decision maker. During negotiation and based on its outcome, a peer can decide if it is better to cooperate or not.
- **Cooperation evaluation:** The interaction is controlled by a TTP, which can be centralized or distributed. In all cases, a conflict regarding negotiation, or remuneration is ultimately arbitrated by the TTP, even though this may possibly be a non immediate process. Deciding about cooperation may be assisted by the authority that can sometimes access to information unavailable to a peer.

These operations can be used repeatedly to perform some cooperative service on a finer granularity basis, which may ease cooperation enforcement. In particular, micropayment is often envisioned rather than an actual (macro-)payment in remuneration based cooperation enforcement mechanisms.

b) Fair exchange

As mentioned in [45], "*many commercial transactions can be modeled as a sequence of exchanges of electronic goods involving two or more parties. An exchange among several parties begins with an understanding about what item each party will contribute to the exchange and what it expects to receive at the end of it. A desirable requirement for exchange is fairness. A fair exchange should guarantee that at the end of the exchange, either each party has received what it expects to receive or no party has received anything*." Fair exchange protocols thus provide ways to ensure that items held by two or more parties are exchanged without one party gaining an advantage. In remuneration systems, obtaining an efficient cooperation incentive depends upon devising a protocol that enforces a fair exchange of the remuneration (virtual or not) against some task. This property can only be attained by intricately integrating the remuneration operation with the application functionality. Fair exchange protocols rely on the availability of a trusted (and neutral) third party (TTP) caring for the correctness of the exchange. Two types of protocols should be distinguished: online protocols, which mediate every interaction through the TTP, which can lead to performance and reliability problems with the TTP constituting a bottleneck as well as a single point of failure; offline ones, also called optimistic fair exchange protocols, which resort to the TTP intermediation only if one the parties wants to prove that the exchange was not fairly conducted.

The TermiNodes project ([15] and [16]) addresses the security of the networking function of packet forwarding through remuneration schemes. Each device possesses a security module that manages its account by maintaining a counter called nuglet, interpreted as virtual money. The project proposes two models for remuneration aiming at enforcing fair exchange for stimulating a cooperative behavior. In the first one, called Packet Purse Model, each packet carries a given number of nuglets and intermediate nodes get paid with some nuglets, which get removed from the packet purse when forwarded by the node. In the second model, called Packet Trade Model, each intermediate node buys packets from the previous node on the route then sells them to the next node for more nuglets, until the destination node, which finally pays the total cost of forwarding packets.

The architecture of Sprite [39], a credit-based system for stimulating cooperation among selfish nodes in mobile ad hoc networks, is not very different from TermiNodes except for the fact that it does not use security modules. It consists of a Credit Clearance Service (CCS) and of mobile nodes. In Sprite, a node transmitting its own messages loses some credits (i.e., virtual money paid by the node to the CCS), which will be used to cover the costs for packet forwarding by intermediate nodes. In order to earn credits, a node must transmit the CCS receipts of forwarded messages. The system does not guarantee balanced payments, i.e., it does not require that the sender’s total debt equal the total intermediate nodes credit received for their forwarding activity. In fact, to prevent cheating behaviors, the CCS debits the sender with a higher amount than that due to intermediate nodes; only later does the CCS uniformly share the exceeding credit among nodes, or give a fixed credit amount to each node. Sprite focuses on combating cheating behaviors and on promoting cooperation among network nodes; it does not prevent active attacks on the system (e.g. Denial of Service attacks).

Remuneration-based mechanisms are also used for peer-to-peer storage like in PAST [30]. PAST is based on the Pastry routing scheme that guarantees that peers contributing to cooperative storage are geographically separated. The storage scheme relies on the use of smart cards to ensure that clients cannot use more remote storage than they are providing locally, which is optional in PAST. Smart cards are held by each PAST user and issued by a third party, and support a quota system that balances supply and demand of storage space in the system. With fixed quotas and expiration dates, users are only allowed to use as much storage as they contribute. In contrast, in OceanStore [12], the remuneration of cooperative peers is monetary as the service is envisioned to be provided by a confederation of companies. In exchange for economic compensation, computers joining the system contribute with storage or provide access to local users. Each user is supposed to pay a fee to one particular provider who buys storage space from and sells it to other providers. Legal contracts and enforcement can be used to punish peers that do not keep their end of the bargain, based on planned billing and auditing systems.

Smart cards (or similar forms of tamper-proof hardware) have been proposed for some time now as a means to implement an optimistic fair exchange protocol. The use of smart cards is especially interesting since it provides both the convenient form factor of a personal token and a perfect implementation of a secure purse. Smart cards also offer a tamper-resistant area for a trusted third party to store secrets, or implement critical security functions, in particular for revoking the user from the system. [44] for instance proposes the use of one card for all parties involved in a transaction and focuses on providing a neutral platform for enforcing fair exchange. Another solution is to use one card for each party [43], which aims at reducing the number of messages exchanged; this solution may also be interesting, although this benefit is not mentioned by the authors, for gaining a better understanding of the liability of each party involved and thus ease the reconciliation of the data by the TTP during the online phase of the protocol, in case of a problem in the offline phase. In addition, the latter technique makes it possible to attach data like some credit to every user and let the smartcard manage this "currency" as part of the fair exchange protocol. As discussed above however, remuneration has to be integrated with the application: this means that interactions with such hardware must be carefully integrated within the application protocol in order to prevent its bypassing or abuse: with smart cards for instance, this involves the mediation of terminals, which are distrusted with respect to remuneration handling.

Table 1 summarizes the various approaches to cooperation and their respective features as discussed in the last two sections.

Network type		Application	Incentives for cooperation		No Incentives for cooperation
			Reputation-based incentives	Remuneration-based incentives	
Decentralized networks	Ad hoc network	Packet forwarding	CORE [32], RPG [7], Watchdog/pathrater [38], CONFIDANT [35]	TermiNodes [15]	
	P2P network	File-sharing	NICE [37]		Napster[13], Gnutella[14], Freenet [11]
		File system	Free Haven [34]	PAST [30], OceanStore [12]	
		Backup	CIBS [20]		Pastiche [17]
De/centralized networks	Mobile network	Backup		FlashBack [5]	
		Info-station			7DS [21]
Centralized networks	Web Sites		Ebay[15], Amazon, Epinions[16], Google[17]		

Table 1 Cooperation enforcement schemes in various applications

Validation techniques

In the context of self-organizing networks like for instance wireless mobile ad hoc networks, cooperative mechanisms have to be investigated in terms of performance, fairness, and resilience to attacks, as well as cooperation enforcement.

7. Simulation

A first validation technique for cooperative systems consists in taking advantage of existing network simulators. These are tailored to fit the simulation context and match the objectives

[13] http://www.napster.com/
[14] http://www.gnutella.com/
[15] http://ebay.com/
[16] http://www.epinions.com/
[17] http://www.google.com/

of simulation by the application of patches and/or individual adjustments. According to [1], "simulation can be defined as the process of designing a model of a real system and conducting experiments with this model for the purpose of understanding the behavior of the system and/or evaluating various strategies for the operation of the system". This means that simulating cooperation incentives with the purpose of testing their efficiency would also require simulating non-cooperative behaviors (and not only an ideal cooperative behavior).

There are some difficulties when simulating overlay networks. Firstly, most overlay networks need to be scalable (thousands of simultaneous users) which is difficult to realize due to memory constraints even for most powerful machines. However, some tools allow a simulation to be distributed over a set of machines (distributed simulation). Additionally, it may be desirable that a simulation behaves in accordance with a physical network (packet delay, traffic and network congestion, bandwidth limitations etc). These considerations also increase the overhead on the host machine. When choosing a network simulation tool for a given network, a distinction must be made between simulation tools that are suitable for low level networks and those that are suitable for overlay networks. Packet-level network simulators such as NS-2 [41], OMNET++ [42], GloMoSim/QualNet ([10], [40]), and OPNET [29] must be distinguished from overlay network simulators like PeerSim [6].

NS-2. There is no doubt that the most popular network simulator is NS (version 2) [41]. NS-2 is an object-oriented, discrete event driven network simulator developed at UC Berkeley. A discrete-event simulator is a simulator where state variables change only at discrete points in time at which events occur caused by activities and delays. The simulator NS-2 is written in C++ and OTcl. NS-2's code source is split between C++ for its core engine and OTcl, an object oriented version of TCL for configuration and simulation scripts. The combination of the two languages offers an interesting compromise between performance and ease of use. An NS-2 simulation scenario is a Tcl file that defines the topology and the movement of each host that participates in an experiment. Implementing a new protocol in NS-2 typically requires adding C++ code for the protocol's functionality, as well as updating key NS-2 OTcl configuration files in order for NS-2 to recognize the new protocol and its default parameters. The C++ code also describes which parameters and methods are to be made available for OTcl scripting. Debugging is difficult in NS-2 due to the dual C++/OTcl nature of the simulator. For the moment, there is only one P2P simulation available for NS2 which is Gnutella. A more troublesome limitation of NS-2 [41] is its large memory footprint and its lack of scalability as soon as simulations of a few hundred to a few thousand of nodes are undertaken. NS-2 is well documented with active mailing lists.

OMNET++. Another discrete event simulator is OMNET++ [42]. OMNET++ is an open-source, component-based simulation environment developed by Andras Varga with a strong focus on supporting the user with a Graphical User Interface (GUI). It is a very modular and well structured simulator. Modules are programmed in C++ and assembled into larger components using a high level language (NED). It is possible to simulate peer-to-peer networks with OMNET++ which can also run distributed simulations over a number of machines. OMNET++ has a rapidly increasing user base now, with lots of useful modules, an active mailing list and even workshops. Both NS-2 and OMNET++ are packet-level simulators; so scalability is a major issue. Thus, both are more suitable for small networks.

GloMoSim/QualNet ([10], [40])**.** GloMoSim is built as a scalable simulation environment for wireless and wired network systems. It is designed using the parallel discrete-event simulation capability provided by PARSEC (C-based simulation language). PARSEC is designed to cleanly separate the description of a simulation model from the underlying simulation protocol, sequential or parallel, used to execute it. GloMoSim is built using a layered approach. Standard APIs are used between the different layers. This allows the rapid integration of models developed at different layers. To specify the network characteristics, the user has to define specific scenarios in text configuration files: app.conf and Config.in. The first contains the description of the traffic to generate (application type, bit rate, etc.) and the second contains the description of the remaining parameters. The statistics collected can be either textual or graphical. With GloMoSim, it is difficult to describe a simple application that bypasses most OSI layers. The bypass of the protocol stack is not obvious to achieve as most applications usually lie on top of it which makes the architecture much less flexible. GloMoSim version is for academic use only. The commercial GloMoSim-based product is QualNet. The development framework is in C/C++ mostly provided in source form. It includes a graphic development tool for adding/revising protocols. There are some similarities between these simulators; specially the fact that they provide as primary function substantial support for simulation of routing protocols over wired and wireless networks.

OPNET. A different commercial tool is OPNET [29] Modeler. The OPNET Modeler is just one of the many tools from the OPNET Technologies suite (Optimized Network Engineering Tools). It is a commercial tool by MIL3, Inc. OPNET is an event-driven scheduled simulator integrating analysis tools for interpreting and synthesizing output data, graphical specification of models and a hierarchical object-based modeling. It can simulate all kinds of wired networks, and an 802.11 compliant MAC layer implementation is also provided. OPNET is a well established product used by large companies to diagnose or reorganize their network. It can simulate wired and wireless networks. Models built with OPNET are hierarchically structured. At the lowest level the process domain is structured as a finite state machine (FSM). The FSM can be structured with the help of a graphical editor that allows the user to specify the relation between the single states and their transitions. The single states and the transition conditions can then be programmed with a C like language called Proto-C. Basically, the deployment process goes through the following phases. First you have to choose and configure the node models you want to use in the simulations, for example, a wireless node, a workstation, a firewall, a router, a web server, etc. Then you build and organize your network by connecting the different entities. The last step consists in selecting the statistics you want to collect during the simulations. Most of the deployment in OPNET is done through a hierarchical GUI. OPNET scales quite well but still there are not enough data in the literature to demonstrate this capability.

PeerSim. In addition to these network simulators, there have also been simulators that only focus on overlay networks. A good example is PeerSim [6] which has been developed for large-scale overlay systems within the BISON project. It takes into account the proprieties of high scalability and dynamism. PeerSim is written in the Java language. It is composed of two simulation engines: the cycle-based one and a more traditional event-based engine. The cycle-based engine does not model the overhead of the transport layer and subsequently is more scalable. The event-based engine is less efficient but more realistic. The simulation engines are supported by many simple, extendable, and pluggable components, with a flexible configuration mechanism.

Other simulations may be found in the existing literature about peer-to-peer or overlay systems e.g. **3LS** [25], **Query-Cycle Simulator** [22], **Anthill** [28] and **NeuroGrid** [27]. None of these simulators is really satisfactory. 3LS, Anthill and NeuroGrid have scalability limitations. Query-Cycle is limited to file-sharing. All lack a sufficient support for dynamicity. In conclusion, we have identified in NS-2 [41], GloMoSim/QualNet ([10], [40]), OMNET++ [42], OPNET [29] and PeerSim [6] the possible candidates to build up the simulation environment for an ad hoc or P2P cooperative system. OPNET and NS-2 possess an extensive set of models, protocols and algorithms already produced, but less than OMNET++. The modular nature of OMNET++ allows the possibility to carry out studies over a wide range of situations in detail. Also, in the sense of the ease of use/modification/extension, OMNET++ appears to be the best simulator. OPNET and QualNet are more than satisfactory with respect to this capability, however NS-2 scores poorly.

Some of the proposed incentives for cooperation schemes were investigated using a network simulator. In 7DS [21] simulation scenarios, hosts were modeled as NS-2 [41] mobile nodes. Mobile nodes move according to the random waypoint mobility model, which is commonly used to model the movement of individual pedestrians. A waypoint model breaks the movement of a mobile node into alternating motion and rest periods. A mobile node moves at a speed uniformly chosen from an interval to a randomly chosen location where it stays for a fixed amount of time; then it chooses another random location and moves towards it, and so on. A NS-2 [41] simulation study was also carried out for the ORION project [3] where the performance of ORION was compared to off-the-shelf approaches based on a P2P file-sharing system for the wired Internet, TCP and a MANET routing protocol. In the simulated scenarios, an IEEE 802.11 standard MAC layer was used along with the standard physical layer, the two-ray ground propagation model. NS-2 was widely employed for simulation principally in wireless mobile networks more than in P2P or ad hoc networks where other simulators like QualNet ([10], [40]) were more or less adopted. CORE [31] was evaluated based on QualNet simulations and CONFIDANT [35] based on GloMoSim ([10], [40]).

	Simulator	P2P protocols	Language	Distributed simulation	Conditions
Packet-level network simulator	NS-2 [41]	Gnutella	C++/OTcl	Yes	Open source
	OMNET++ [42]	None	C++/NED	Yes	Academic public license
	GlomoSim/ QualNet ([10], [40])	---	C/C++	Yes	Free for universities / commercial
	OPNET [29]	---	Proto-C	Yes	Commercial
Overlay network simulator	PeerSim [6]	Collection of internally developed P2P models	Java	No	Free
	3LS [25]	Gnutella	Java	No	---
	NeuroGrid [27]	Gnutella, NeuroGrid, Pastry, FreeNet [11]	Java	No	Free

Table 2 Characteristics of discussed simulators [1]

To validate the cooperative system, it is possible to simulate the system with a simulation model created for this purpose without resorting to an existing network simulator. A mobile P2P file-sharing simulation model in [14] contains a "network" component to model the network and devices' particularities and restrictions. It includes as well a "source traffic" component to model the transmitted data and the behavior of peers in the network. The proposed mobile P2P architecture in this model is based on the eDonkey P2P file-sharing protocol and is enhanced by additional caching entities and a crawler. In the simulation, a mobile peer is described by an ON/OFF-process to reflect the fluctuating connection status of a mobile peer. ON and OFF periods are determined by exponential distributions. The file request arrivals are modeled by a Poisson process. An abstract model using a subset of the parameters of the detailed simulation is proposed to reduce the computing time. The abstract model is used to identify which cache replacement strategy fits the best for the mobile P2P system. For this, the request arrival process is simulated in detail while the used transport mechanism and the upload queue mechanism are neglected.

8. Game theory

Results obtained through simulation studies give a proof-of-concept of the proposed cooperative mechanism. The results do not demonstrate if the incentives for cooperation are crucial. For this, game theory provides an alternative tool to declare that a cooperative mechanism is a cooperation strategy. Game theory models strategic decision situations where self-interested users follow a strategy aiming at maximizing their benefits and minimizing their resource consumption. Game theory offers different methods for study e.g. non-cooperative game, cooperative game, and evolutionary game. *Non-cooperative game* focuses on users' strategies. It describes the strategy of a user that has to make a decision about whether to cooperate or not with a randomly chosen user. On the other hand, *cooperative game* focuses on mutually advantageous results for the different parties. In this game, users are able to enforce contracts and make binding agreements. *Evolutionary games* study the evolution of various strategy profiles over time and space.

We describe the decision making process that a peer will undertake when participating in a non-cooperative game, through the example of a classical game, *the prisoner's dilemma*. In this well-known game, two players are both faced with a decision to either cooperate (C) or defect (D). The two players' decisions are made simultaneously with no knowledge of the each other's decision. If the two players cooperate they receive a benefit R. If both defect they receive a punishment P. If one player defects and the other one cooperates, the defecting player receives a given benefit T and the cooperator a punishment S. The canonical form of the prisoner's dilemma pay-off is shown in the table below:

		Player 2	
		C	D
Player 1	C	(R, R)	(S, T)
	D	(T, S)	(P, P)

Table 3 Prisoner's dilemma pay-off matrix

In order to have a dilemma, the following expressions must hold:

$$T>R>P>S \text{ and } R>(S+T)/2$$

We see here that a player in this dilemma has better to defect regardless of the decision of the other player because the strategy D strictly dominates the strategy C (T>R and P>S). The solution of this game is the *Nash equilibrium*. We call a set of strategies a Nash equilibrium in the case where no player could improve his payoff, given the strategies of all other players in the game, by changing his strategy while the other players keep their strategies unchanged. The analysis of the interaction between decision-makers involved in the prisoner's dilemma game can be extended to *repeated (or iterated) games*. There are several strategies that a player can adopt to determine whether to cooperate or not at each of its moves in the repeated game. There is a strategy known as tit-for-tat where a player cooperates in the first period then copy his opponent's last move for all subsequent periods. Another strategy called Spiteful is to cooperate in the first period and for later periods cooperate if both players have always cooperated; if either player defects then defect for the remainder of the game. A cooperation enforcement mechanism can be translated into a strategy for a player and compared to these straightforward strategies. Another important concept is the idea of *evolutionary stable strategy*. A set of strategies is at evolutionary stable strategy equilibrium if no individual playing one strategy could improve its payoff by switching to one of the other strategies in the proportion and also no individual playing a different strategy (called a mutant) could establish itself in (invade) the population, i.e., make other individuals in the population choose his strategy. With these different concepts we can give a good analysis of the cooperation enforcement mechanism in terms of both promoting cooperation and the evolution of cooperation. Cooperation and coalition formation can be explained using a two-period structure. Players first decide whether or not join a coalition and in the second step the coalition and non-cooperative peers choose their behavior non-cooperatively. A coalition is defined as stable if no peer in the coalition has an incentive to leave. In this sense, a preference structure was suggested defined by the *ERC-theory* [9]. In this theory, the utility of a decision-maker is not solely based on the absolute payoff but also on the relative payoff compared to the overall payoff to all peers. The model explains observations from games where equity, reciprocity or competition plays a role.

The CORE mechanism [31] was for instance validated following two methodologies. The first approach was to use a simulation tool, GloMoSim ([10], [40]), while the second validation used a game-theoretic methodology. For this, two models were provided based on cooperative and non-cooperative game theory to demonstrate the need for cooperation incentives in the network. The CORE mechanism was then translated into a strategy model and its evolutionary stability was proven. Further, the authors showed that in a more realistic scenario (communication errors, failures etc) CORE outperforms other basic cooperation strategies. Sprite [39] also used a game-theoretic model to prove that the solution of the scheme prevents cheating behaviors. The main results work for packet forwarding in unicast communication. The most important role of the game-theoretic validation of Sprite algorithms is in determining payments and charges of nodes in the system to motivate each node to cooperate honestly and to report its behavior to the CCS (Credit Clearance Service). Finally, remuneration fairness was studied from a game theoretic point of view in [46], in which the authors discuss different equilibrium concepts as a model for the various types of fair exchange.

9. Prototype-based evaluation

A cooperation mechanism can be validated just a stage before the final design by building a prototype, that is, a physical model of a proposed product concept that allows demonstration, evaluation, or testing of the product. A prototype thus combines the most representative attributes and particularities of a mechanism and it can be developed for testing it. It is also usually intended to evolve the design into a more final state. In the literature, there are a lot of examples of P2P or ad hoc cooperation mechanisms where the evaluation process is based on prototypes. To validate their incentives system for ad hoc networking, a prototype was used for Sprite [39] to determine how much overhead the incentive scheme necessities and how is the packet routing performance of the system (percentage of packets successfully relayed from the sender to the destination). Results show that the overhead of the Sprite system is insignificant, and that nodes in the system cooperate and forward each other’s messages unless their resources are extremely low. The Pastiche [17] scheme was evaluated using also the prototyping approach. Pastiche's prototype consists of two main components: the chunkstore file system implemented in user space and written in C and a backup daemon. With the evaluation of the prototype, it was demonstrated that the backup service does not penalize the file system performance unduly and also that node discovery was effective. CIBS [20] was also prototyped for the validation of the backup scheme. To measure the performance of their Internet backup scheme its authors used a number of personal computers running instances of the prototype software. Each instance was partnered with the other instances located in different PCs so that all communication between partners went through the network. Experiments on the prototype have shown that the backup scheme performance is acceptable in practice and that the technique is feasible and cheap.

Conclusions

Different approaches can be taken for cooperation enforcement, yet cooperation evaluation is clearly the part most dependent on application-specific requirements and constraints, in particular concerning deployment. The choice of a particular technique for validating the effectiveness of cooperation, a critical step to ensuring that the application will reach its objectives, depends heavily on the application chosen. This may in particular hamper the use of one technique because of scalability issues or of the properties that need to be proven.

Trust, as one would like to evaluate it in the applications mentioned above, can be static (identity-based for instance) or dynamic (self-organized). Static trust refers to the statement of trust that remains the same until it is revoked, whereas dynamic trust exhibits self-learning and self-amplifying characteristics. The latter arises from behaviors experienced in the system and continuously changes accordingly to them. An entity trusts another entity because the information it has about it shows that the actual entity is trustful and also because it has a lot of information about it. [19] for instance introduces a trust model that does not only concentrate on the content of evidence but also on the amount of such evidence.

Trust, although closely related with cooperation, may not be valuably accounted for by all cooperation evaluation metrics of the mechanisms listed above. In particular, whereas reputation seems well adapted to reason with the trustworthiness of a peer, remuneration may be much poorer semantically, especially if the payment may be used to enforce cooperation for different self-organized services. This does not mean that trust does not require cooperation as a prerequisite, but instead that trust establishment might not be as reliable as expected if the evaluation of its cooperative component relies on an unsuitable incentive mechanism.

References

[1] A. Brown, M. Kolberg, “Tools for Peer-to-Peer Network Simulation”, March 3rd, 2006, http://www.ietf.org/internet-drafts/draft-irtf-p2prg-core-simulators-00.txt

[2] A. Jøsang, R. Ismail, and C. Boyd, “A Survey of Trust and Reputation Systems for Online Service Provision”, In Proceedings of *Decision Support Systems,* 2005.

[3] A. Klemm, C. Lindemann, and O. Waldhorst, “A Special-Purpose Peer-to-Peer File Sharing System for Mobile Ad Hoc Networks”, *Proc. IEEE Semiannual Vehicular Technology Conference (VTC2003-Fall)*, Orlando, FL, October 2003.

[4] A. Montresor, G. Di Caro, P. E. Heegaard, “Architecture of the Simulation Environment”, BISON IST-2001-38923, January 29, 2004.

[5] B. T. Loo, A. LaMarca, and G. Borriello, “Peer-To-Peer Backup for Personal Area Networks”, Intel Research Technical Report IRS-TR-02-15, October 2002.

[6] Biology-Inspired techniques for Self-Organization in dynamic Networks, BISON Project, http://www.cs.unibo.it/bison/

[7] D. Barreto, Y. Liu, J. Pan, and F. Wang, “Reputation-based participation enforcement for ad hoc networks”, 2002.

[8] DATAMAN, http://planchet.rutgers.edu/~badri/dataman/research-projects.html

[9] G. E Bolton and A. Ockenfels, “ERC: a theory of equity, reciprocity, and competition”, *American Economic Review 90(1): 166-193*, 2000.

[10] Global Mobile Information Systems Simulation Library, GloMoSim, http://pcl.cs.ucla.edu/projects/glomosim/

[11] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong, “Freenet: A distributed anonymous information storage and retrieval system”, In Designing Privacy Enhancing Technologies: International Workshop on Design Issues inAnonymity and Unobservability, LNCS 2009, New York, 2001.

[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 globalscale persistent storage”, *in Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, Nov. 2000.

[13] J. Liang, R. Kumar & K.W. Ross, “The FastTrack overlay: A measurement study”, *Computer Networks, 50*, 842-858, 2006.

[14] J. Oberender, F. –U. Andersen, H. de Meer, I. Dedinski, T. Hoßfeld, C. Kappler, A. Mäder, and K. Tutschku, “Enabling Mobile Peer-to-Peer Networking. Mobile and Wireless Systems”, In *Proceedings of Mobile and Wireless Systems, LNCS 3427,* Dagstuhl, Germany, January 2005.

[15] L. Buttyán and J. Hubaux, “Nuglets: a virtual currency to stimulate cooperation in self-organized ad hoc networks”, Technical report, EPFL, 2001.

[16] L. Buttyán and J.-P Hubaux, “Stimulating Cooperation in Self-Organizing Mobile Ad Hoc Networks”, *ACM/Kluwer Mobile Networks and Applications*, 8(5), October 2003.

[17] L. P. Cox and B. D. Noble, “Pastiche: making backup cheap and easy”, in Proceedings of the Fifth USENIX Symposium on Operating Systems Design and Implementation, Boston, MA, December 2002.

[18] L. Page, S. Brin, R. Motwani, and T. Winograd, “The PageRank Citation Ranking: Bringing Order to the Web”, Technical report, Stanford Digital Library Technologies Project, 1998.

[19] M. Carbone, M. Nielsen, and V. Sassone, “A Formal Model for Trust in Dynamic Networks”, BRICS *tech. report RS-03-4*, Univ. Aarhus, 2003.

[20] M. Lillibridge, S. Elnikety, A. Birrell, M.Burrows, and M. Isard, “A Cooperative Internet Backup Scheme”, *In Proceedings of the 2003 Usenix Annual Technical Conference (General Track), pp. 29-41*, San Antonio, Texas, June 2003.

[21] M. Papadopouli and H. Schulzrinne, “A Performance Analysis of 7DS, A Peer-to-Peer Data Dissemination and Prefetching Tool for Mobile Users”, In Advances in wired and wireless communications, IEEE Sarnoff Symposium Digest, March 2001.

[22] M. Schlosser and S. Kamvar, “Simulating a file-sharing p2p network”, In *Proceedings of the First Workshop on Semantics in P2P and Grid Computing*, 2002.

[23] Markus Jakobsson, Jean-Pierre Hubaux, and Levente Buttyan, “A Micro-Payment Scheme Encouraging Collaboration in Multi-Hop Cellular Networks”, In Proceedings of Financial Crypto, La Guadeloupe, Jan. 2003.

[24] N. Asokan, M. Schunter, and M. Waidner. Optimistic Protocols for Fair Exchange. In Proceedings of the 4th ACM Conference on Computer and Communications Security, Zurich, April 1997.

[25] N. S. Ting, R. Deters, “3LS - A Peer-to-Peer Network Simulator”, *IEEE International Conference on Peer-to-Peer Computing*, 2003.

[26] Nadia Ben Azzouna and Fabrice Guillemin, “Experimental analysis of the impact of peer-to-peer applications on traffic in commercial IP networks”, European transactions on Telecommunications: Special issue on P2P networking and P2P services, ETT 15(6), November-December 2004.

[27] NeuroGrid, http://www.neurogrid.net

[28] O. Babaoglu, H. Meling, and A. Montresor, “Anthill: A framework for the development of agent-based peer-to-peer systems”, In *Proceedings of the 22th International Conference on Distributed Computing Systems (ICDCS'02)*, Vienna, Austria, July 2002.

[29] OPNET, http://www.opnet.com/

[30] P. Druschel and A. Rowstron, “PAST: A large-scale, persistent peer-to-peer storage utility”, *in Proceedings of HotOS VIII*, May 2001.

[31] P. Michiardi, “Cooperation enforcement and network security mechanisms for mobile ad hoc networks”, PhD Thesis, December 14th, 2004.

[32] P. Michiardi, and R. Molva, “CORE: a collaborative reputation mechanism to enforce node cooperation in mobile ad hoc networks”, *CMS'2002, Communication and Multimedia Security 2002 Conference*, Portoroz, Slovenia, September 26-27, 2002.

[33] P. Obreiter & J. Nimis, “A Taxonomy of Incentive Patterns - the Design Space of Incentives for Cooperation”, Technical Report, Universität Karlsruhe, Faculty of Informatics, 2003.

[34] R. Dingledine, “The free haven project: Design and deployment of an anonymous secure data haven”, *Master's thesis, MIT*, June 2000.

[35] S. Buchegger, and J. Y. L. Boudec, “Performance Analysis of the CONFIDANT Protocol: Cooperation Of Nodes — Fairness In Distributed Ad-hoc NeTworks”, *In Proceedings of IEEE/ACM Workshop on Mobile Ad Hoc Networking and Computing (MobiHOC), Lausanne, CH, IEEE (2002) 226–236*, 2002.

[36] S. D. Kamvar, M. T. Schlosser, and H. Garcia-Molina, “The EigenTrust Algorithm for Reputation Management in P2P Networks”, In Proceedings *of the Twelfth International World Wide Web Conference*, Budapest, May 2003.

[37] S. Lee, R. Sherwood, B. Bhattacharjee. "Cooperative peer groups in NICE". In INFOCOM'03, April 2003.

[38] S. Marti, T.J. Giuli, K. Lai, and M. Baker, “Mitigating routing misbehavior in mobile ad hoc networks”, *Mobile Computing and Networking 255–265*, 2000.

[39] S. Zhong, J. Chen, Yang Richard Yang, “Sprite: A Simple, Cheat-Proof, Credit-Based System for Mobile Ad-Hoc Networks”, INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies.

[40] Scalable Network Technologies (SNT), http://www.scalable-networks.com/products/

[41] The network Simulator NS-2, http://www.isi.edu/nsnam/ns/index.html

[42] The OMNeT++ Community Site, http://www.omnetpp.org/

[43] M. Terada, M. Iguchi, M. Hanadate, and K. Fujimura. An Optimistic Fair Exchange Protocol for Trading Electronic Rights. In 6th Smart Card Research and Advanced Application conference (CARDIS'2004), 2004.

[44] H. Vogt, H. Pagnia, and F. C. Gärtner. Using Smart cards for Fair-Exchange. WELCOM 2001, LNCS 2232, Springer, pp. 101-113, 2001. http://citeseer.ist.psu.edu/vogt01using.html

[45] N. Asokan, M. Schunter, and M. Waidner. Optimistic Protocols for Fair Exchange. In Proceedings of 4th ACM Conference on Computer and Communications Security, Zurich, April 1997. http://citeseer.ist.psu.edu/article/asokan96optimistic.html

[46] L. Buttyan and J.-P. Hubaux, Toward a formal model of fair exchange -- a game theoretic approach, 2000, http://citeseer.ist.psu.edu/article/buttyan00toward.html, SSC Technical Report

4-Published Papers

Collaborative Backup for Dependable Mobile Applications

[Extended Abstract]

Marc-Olivier Killijian, David Powell
LAAS-CNRS
7 Avenue du Colonel Roche
31077 Toulouse cedex 4
France

Michel Banâtre, Paul Couderc
IRISA
Campus Universitaire de Beaulieu
35042 Rennes cedex, France

Yves Roudier
Institut Eurécom
2229 Route des Crêtes
Sophia Antipolis
06560 Valbonne
France

ABSTRACT

We describe the work we are conducting on new middleware services for dependable and secure mobile systems. This work is based on approaches à la peer-to-peer in order to circumvent the problems introduced by the lack of infrastructure in self-organizing networks of mobile nodes, such as MANETs. The mechanisms we propose are based on collaboration between peer mobile devices to provide middleware services such as trust management and critical data storage. This short paper gives a brief description of the problems we are trying to solve and some hints and ideas towards a solution.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems

Keywords

Mobile applications, data back-up, collaboration

1. INTRODUCTION

The MoSAIC (Mobile System Availability, Integrity and Confidentiality) project [9] aims to investigate novel dependability and security mechanisms for mobile wireless devices, especially personal mobile devices, in ambient intelligence applications. The mobile devices of interest include, for instance: personal digital assistants (PDAs), laptop computers, mobile telephones, digital cameras, etc., and extend to systems embedded within vehicles. The focus is on sparse ephemeral self-organizing networks, using predominately single-hop wireless communication, i.e., networks of a small number of a potentially large population of mobile devices that come into existence spontaneously by virtue of physical proximity and mutual discovery, and that cease to exist as soon as communication is no longer possible.

Most of the data carried on a PDA is a copy of data that is mainly produced and also stored elsewhere. For example, a PDA contact database is regularly synchronized with a desktop computer application. This reduces the impact of failure of such devices to the data that is produced directly on the device between synchronizations. However, in the case of capture devices (devices capable of acquiring data such as pictures, sound or video), large quantities of data are generated directly on the mobile device, leading to a much larger quantity of data that remains sensitive to device failure until a backup copy can be created. This highlights the need for new ways of ensuring data availability. Because the density of these devices is increasing (as mobile devices are becoming more and more popular), there is an opportunity for cooperatively backing up data by using neighborhood devices. The first objective of our work is therefore to define an automatic data back-up and recovery service based on mutual cooperation between mobile devices with no prior trust relationships. Such a service aims to ensure continuous availability of critical data managed by mobile devices that are particularly prone to energy depletion, physical damage, loss or theft. The basic idea is to allow a mobile device to exploit accessible peer devices to manage backups of its critical data. To our knowledge, no work has already exploited this principle of cooperative backup for mobile devices. Indeed, relatively little work appears to have been devoted to tolerance of device failures in a mobile self-organized network scenario [14] [2] [1], although there has been considerable work on checkpointing in cellular mobile computing environments (see, e.g., [18] [19] [20] [4] [16] [17]).

The implementation of such a service by cooperation between mobile nodes with no prior trust relationship is far from trivial since new threats are introduced: (a) selfish devices may refuse to cooperate; (b) backup repository devices may themselves fail or attack the confidentiality or integrity of the backup data; (c) rogue devices may seek to deny service to peer devices by flooding them with fake backup requests; etc. We intend to study trust management mechanisms to support cooperative services between mutually suspicious devices. Of particular interest are mechanisms based on reputation (for prior confidence-rating and posterior accountability) and rewards (for cooperation inci-

2nd Workshop on Middleware for Pervasive and Ad-Hoc Computing Toronto, Canada
Copyright 2004 ACM 1-58113-951-9 ...$5.00.

tation). In the sparse ephemeral networks considered, these mechanisms can rely neither on accessibility to trusted third parties nor on connectivity of a majority of the considered population of devices [22]. Self-carried reputation and rewards are therefore of prime interest. This approach contrasts to most existing approaches to mobile system security, which have mainly focused on key management and distribution (see, e.g.,[22] [8] [12]) and on secure ad-hoc network routing (see, e.g., [3] [6] [15] [21]).

Achieving dependability and security despite accidental and malicious faults in networks of mobile devices is particularly challenging due to their intrinsic asynchrony (unreliable communication, partitioning, mobility, etc.) and the consequent absence of continuous connectivity to global resources such as certification and authorization servers, system wide stable storage, a global time reference, etc. Furthermore, the threats to dependability and security are particularly severe: device lifetime and communication are severely limited by scarcity of electrical energy; use of wireless links means susceptibility to link attacks ranging from passive eavesdropping to active impersonation, message replay, and message distortion; poor physical protection of mobile devices (especially in a hostile environment) makes them susceptible to physical damage, and vulnerable to theft or subversion.

There are thus two related issues that need to be addressed :

1. Fault- and intrusion-tolerant collaborative data backup (with possible extension to checkpointing).

2. Self-carried reputation and rewards for collaboration between sporadically interconnected and mutually suspicious peer devices without reliance on a fixed infrastructure and access to trusted third parties.

Common to both is our emphasis on spontaneous interaction between peer mobile devices with no prior trust relationships. In this paper, we focus on the first of these two issues.

2. FAULT TOLERANCE BY COLLABORATIVE BACKUP

We are investigating middleware services to support the dependability and security of mobile ambient intelligence applications. We consider highly dynamic systems consisting of wireless-equipped mobile devices that communicate with each other mostly by direct, single-hop communication. However, we do not preclude extensions to include indirect communication via a multi-hop ad-hoc network or occasional access to a fixed communication infrastructure. We are not addressing mobile ad-hoc routing protocols, or dependability and security issues at the wireless network level, which are largely covered in the litterature.

We consider the design and implementation of a prototype service for data backup and recovery by cooperation between ephemerally-connected and mutually-suspicious mobile devices. The problems we consider arise from the specific characteristics of ambient intelligence applications based predominately on sparse ephemeral networks of mobile devices: disconnected mode or absence of fixed infrastructure, absence of prior organization, ephemeral interactions, user transparency, and user privacy. Limits on mobile device energy, computation and storage will also constrain the technical solutions that can be considered.

A typical scenario for such a service might be a researcher travelling to a conference, using her PDA to take important notes, and using the PDAs of fellow attendees or travellers to host temporary back-ups of critical data. Such temporary back-ups provide the means for recoverying critical data in the event that her PDA fail, break or be stolen. Recovery can be achieved by purchasing a new PDA, authenticating it and then recollecting the critical data chunks backed-up on other devices. An important aspect of this scenario concerns the fact that the users (and their devices) cooperating for achieving this back-up service have no prior trust relationship. They must thus protect, for example (a) the backed-up data against confidentiality and availability attacks and (b) the back-up devices against denial of service attacks.

Other scenarios, with varying prior trust models, can be imagined in military applications (e.g., recovery and redistribution of critical command and control data during battlefield operations), civilian emergency operations, home automation and entertainment, etc.

The need for such a fault-tolerance service is motivated by: (a) the increasing dependency of users on the availability, integrity and confidentiality of data carried by mobile devices and (b) the fragility of mobile devices and other risks relating to their use in a harsh or even hostile environment. We purposely limit ourselves to the issue of data backup, but note that such a service could serve as the basis for mobile device checkpointing and recovery, and for real-time tolerance of mobile device failure based on redundant devices.

The problems to be addressed include: resource allocation, garbage collection of obsolete backups, integrity and confidentially of backup data, resistance to denial-of-service (DoS) attacks, etc. The service is to be supported by negotiation between peer mobile devices with no prior trust relationship. Among the various approaches that might be considered, we intend to take inspiration from current work in the area of peer-to-peer (P2P) applications [13] [5] [10] [11], which have characteristics that are particularly well-adapted to the considered environment: absence of pre-established organization, service through cooperation, short-duration interactions, etc. We also plan to take inspiration from our know-how in the domain of fragmentation-replication-dissemination (FRD) techniques, which exploit distribution to increase availability, integrity and confidentiality in the face of accidental faults and malicious attacks [7]. Until now, these FRD techniques have only been considered in the case of fixed infrastructure systems. We might also consider the advantages that could be drawn from occasional access to a common time reference (e.g., through the Global Positioning System (GPS)) or from exploiting mobility for data dissemination.

In the sequel, we use the terms *data owner* to refer to a device requesting its data to be backed up and *data saver* for a device hosting back-up data. Any device may be both a data owner and a data saver. However, to simplify our discourse, we usually consider a single data owner.

2.1 Threats

The data back-up service must face up to the following threats:

1. Permanent and transient accidental faults affecting a

data owner.

2. Theft or loss of a data owner device.

3. Accidental or malicious faults causing a data saver to be unavailable when recovery is required (i.e., on failure of the data owner).

4. Accidental or malicious modification of data backups that could violate data integrity if recovery should be required.

5. Malicious read access to data backups. Back-ups may contain sensitive confidential data that should be made unintelligible to the user of the data saver device.

6. Denial of service through selfishness. Cooperation may be thwarted if there is no incentive for devices to participate.

7. Denial of service through maliciousness. A malicious data owner could attempt to saturate data savers by false back-up requests, and thereby deny service to other data owners and to users of the attacked data saver devices. A malicious data saver may also choose to withhold backed-up data (cf. threat 3).

It will also be important to distinguish various contexts of utilization of the data back-up service according to the type of user community and appropriate prior trust model. For example, in a closed (and non-infiltrated) military context, certain threats such as denial-of-service through selfishness or malicious attack may be considered negligible.

2.2 Back-up process

The primary aim of the back-up service is to provide protection against permanent and transient accidental faults of data owners (threat 1). Depending on the utilization context, complete or partial back-up of data may be considered. Partial *delta* back-ups or update operation logs might be preferred to minimize the amount of data to be transferred to and stored on data savers, or even to provide some protection against confidentiality attacks on back-ups (threat 5).

The back-up service also provides protection of data availability in the face of loss or theft of the data owner device (threat 2). Confidentiality might be provided in such a situation by an "auto-delete" function triggered by a failed user-authentication challenge.

Unavailability and modification of back-ups (threats 3 and 4) are only of importance if the data owner should fail. Tolerance of multiple faults may be achieved by installing redundant back-ups on independent data savers. Malicious read access to back-ups (threat 5) may be prevented by cryptographic techniques, with appropriate trade-offs between the level of protection provided and the associated costs in energy and resource consumption. The strength (key length, degree of redundancy, etc.) and cost of the deployed techniques may be adapted according to the degree to which data savers may be trusted (e.g., devices of colleagues or those of strangers). The adaptation could also make use of a dynamic measure of the "reputation" of the data saver (cf. issue 2 raised in the introduction).

Fragmentation–replication–dissemination (FRD) techniques [7] are also of interest here. Data confidentiality may be provided by cutting back-up data into fragments that are disseminated over different data savers. Fragments may also be replicated to ensure data availability and integrity (by voting on multiple replicas). Fragmentation, replication and dissemination may be modulated in both space and time according to the number of trustable devices available in a given place or at a given instant.

Denial of service through selfishness (threat 6) may be discouraged by the use of a "reward" scheme to motivate device participation, inspired from micro-economy approaches developed in peer-to-peer applications. Devices acting as data savers are rewarded for their participation and may redeem their earnings when acting as data owners that wish to purchase back-up service. Denial of service through maliciousness (threat 7) may also be discouraged by an appropriate "reputation" mechanism. Devices with a history of detected maliciousness will have a poor reputation and will be spurned by data owners when negotiating to purchase back-up service. The related notions of reward and reputation are the subject of the cooperative service trust mechanisms that we also plan to investigate.

2.3 Recovery process

The second important aspect of the proposed data back-up service concerns the means by which back-up data may be re-installed when required on data owners, i.e., data recovery. This involves finding the data that has been backed up and transferring it back to the data owner or its surrogate.

The recovery process will depend heavily on whether or not devices can occasionally connect to a fixed infrastructure. If access to a fixed infrastructure cannot be considered (e.g., in a battlefield scenario), then access to back-up data has to be based on establishing a wireless communication channel between data owner and saver devices. If direct communication is not possible (which will be the usual case) then the solution may be to create an ad-hoc network with intermediate devices, or to wait until the devices are again within wireless range (by chance encounter or by planned rendezvous).

At least two recovery modes can be distinguished:

- *Push* recovery: the data saver automatically sends data backups to the data owner or its surrogate. The most appropriate way might be for data savers to trigger such a boomerang operation as soon as they have access to a fixed infrastructure. The data could be transferred either immediately to the data owner or its surrogate, or possibly through a trusted third party.

- *Pull* recovery: the data owner searches for the data copies that it requires. Again, we may take inspiration from P2P systems that seek to develop totally distributed file search engines. Requests to the search engine might target the requested data by specifying particular places or times, e.g., "the data I backed up during the flight from Toulouse to Rennes on January 10, 2004".

When partial back-ups have been created, like when fragmentation-replication-dissemination is used, the recovery process will also need to tackle the problem of reconstructing the complete data from the various parts.

Many various optimizations of the proposed back-up service may be considered. For example, in the case of incremental back-ups, the optimal period of back-up creation may depend on several factors, including the relative size of the increments (deltas or update logs) and the performance of recovery based on those increments. The chosen solutions need to be flexible and adaptable to various application scenarios. Another important issue is that of garbage-collecting obsolete back-up data. This may depend on the notion of a contract set up between data owners and savers, or be triggered when the data owner announces that the earlier back-ups are obsolete. The appropriate solutions imply various business models associated with micro-economy mechanisms of various complexity: fines, contracts, leases, etc.

3. CONCLUSIONS

While the area of dependability of the low level network layers for mobile devices has received much attention (e.g. fault-tolerant routing), middleware and application-level dependability mechanisms remain almost unexplored. As mobile devices become more and more common - we can now embed a real-time operating system with wireless capabilities in a wrist-watch - users will increasingly use them for more critical tasks and will expect greater reliability from them. For example, loosing the automatically gathered orders of the clients that a salesman visited during the morning is completely unacceptable. Even if most of the data carried on a PDA is typically regularly synchronized with a desktop computer, some of its data is produced or modified between these synchronizations. In the case of capture devices, this amount of data is even larger. The user cannot afford to lose the critical data created or modified between synchronizations. The mechanisms we describe in this paper try to tackle the issue of using peer-provided resources for building a collaborative backup service between mobile devices with no prior trust relationship. We think that the impact of such a technology will be high and can be extended to other scenarios like exploratory operations, sensor networks and military missions.

4. REFERENCES

[1] M. Boulkenafed and V. Issarny. AdHocFS: Sharing Files in WLANs. In *2nd Int. Symp. on Network Computing and Applications*, pages 156–63. IEEE CS Press, 2003.

[2] M. Boulkenafed and V. Issarny. A middleware service for mobile ad hoc data sharing, enhancing data availability. In *4th ACM/IFIP/USENIX International Middleware Conference*, pages 493–511. Springer, 2003.

[3] S. Buchegger and J.-Y. L. Boudec. The selfish node: Increasing routing security in mobile ad hoc networks. Technical Report RR 3354, IBM, May 2001.

[4] G. Cao and M. Singhal. Mutable checkpoints: a new checkpointing approach for mobile computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 12:157–72, 2001.

[5] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A distributed anonymous information storage and retrieval system. *Lecture Notes in Computer Science*, 2009:46, 2001. `http://freenet.sourceforge.net`.

[6] B. Dahill, B. Levine, E. Royer, and C. Shields. A secure routing protocol for ad hoc networks. In *10th Conference on Network Protocols (ICNP)*, November 2002.

[7] Y. Deswarte, L. Blain, and J.-C. Fabre. Intrusion tolerance in distributed systems. In *IEEE Symposium on Security and Privacy*, pages 110–121. IEEE CS Press, 1991.

[8] A. Khalili, J. Katz, and W. A. Arbaugh. Toward secure key distribution in truly ad-hoc networks. In *Symp. on Applications and the Internet Workshops (SAINT'03 Workshops)*, pages 342–46, 2003.

[9] M.-O. Killijian, M. Banâtre, P. Couderc, L. Courtès, S. Crosta, R. Molva, D. Powell, Y. Roudier, and F. Weiss. The MoSAIC project. `http://www.laas.fr/mosaic/`.

[10] D. Kügler. An analysis of gnunet and the implications for anonymous, censorship-resistant networks. `http://www.ovmj.org/GNUnet/`.

[11] S. Lee, R. Sherwood, and B. Bhattacharjee. Cooperative peer groups in NICE. In *INFOCOM'03*, April 2003.

[12] D. Liu, P. Ning, and K. Sun. Efficient self-healing group key distribution with revocation capability. In *10th ACM Conf. on Computer and Communications Security (CCS'03)*, pages 231–40, 2003.

[13] MNET. The MNET project. `http://mnetproject.org`.

[14] P. Nikander. Fault tolerance in decentralized and loosely coupled systems. In *Ericsson Conference on Software Engineering.* Ericsson, 2000.

[15] P. Papadimitratos and Z. J. Haas. Secure routing for mobile ad hoc networks. In *SCS Communication Networks and Distributed Systems Modeling and Simulation Conference (CNDS 2002)*, 2002.

[16] T. Park, N. Woo, and H. Y. Yeom. An efficient recovery scheme for mobile computing environments. In *Int. Conf. on Parallel And Distributed Systems (ICPADS)*, pages 53–60. IEEE CS Press, 2001.

[17] C. Pedregal-Martin and K. Ramamrithan. Support for recovery in mobile systems. *IEEE Transactions of Computers*, 51:1219–24, 2002.

[18] D. K. Pradhan, P. Krishna, and N. H. Vaidya. Recoverable mobile environment: Design and trade-off analysis. In *26th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-26)*, pages 16–25. IEEE CS Press, 1996.

[19] R. Prakash and M. Singhal. Low-cost checkpointing and failure recovery in mobile computing systems. *IEEE Transactions on Parallel and Distributed Systems*, 7:1035–48, 1996.

[20] B. Yao, K.-F. Ssu, and W. K. Fuchs. Message logging in mobile computing. In *29th IEEE Int. Symp. on Fault-Tolerant Computing (FTCS-29)*, pages 294–301. IEEE CS Press, 1999.

[21] M. Zapata and N. Asokan. Securing ad hoc routing protocols. In *ACM Workshop on Wireless Security (WiSe 2002)*, September 2002.

[22] L. Zhou and Z. Haas. Securing ad hoc networks. *IEEE Network Magazine*, 13:24–30, 1999.

Sauvegarde coopérative entre pairs pour dispositifs mobiles[*]

Ludovic Courtès Marc-Olivier Killijian David Powell Matthieu Roy

prénom.nom@laas.fr
LAAS-CNRS
7 Avenue du Colonel Roche
31077 Toulouse cedex 4
France

Résumé/Summary

Nous présentons les fonctionnalités d'un service de sauvegarde coopérative pour dispositifs mobiles. Ce service repose sur la collaboration entre dispositifs pour assurer la sauvegarde et le recouvrement des données de chaque dispositif. Nous identifions les propriétés de sûreté de fonctionnement que l'on est en mesure d'attendre d'un tel service. Nous analysons les systèmes de sauvegarde coopérative pair-à-pair décrits dans la littérature afin d'identifier d'éventuelles fonctionnalités transposables à l'environnement mobile. Enfin, nous concluons sur les spécificités de cet environnement et identifions les axes de recherche à explorer.

Mots-clef : Sauvegarde, dispositifs mobiles, coopération, pair-à-pair.

We present the features of a collaborative backup service for mobile devices. This service relies on collaboration among peers in order to provide data backup and recovery. We identify the expected dependability properties for such a service. We then survey peer-to-peer collaborative backup systems described in the literature and identify mechanisms relevant to the mobile environment. We conclude on the specificities of this environment and identify future research directions.

Keywords : Mobile applications, data back-up, collaboration, peer-to-peer.

[*] Ces travaux s'intègrent dans le cadre du projet MoSAIC (*Mobile System Availability, Integrity and Confidentiality, http://www.laas.fr/mosaic/*) partiellement financé par l'Action Concertée Incitative (ACI) Sécurité & Informatique du Ministère délégué à la Recherche et aux Nouvelles Technologies.

Sauvegarde coopérative entre pairs pour dispositifs mobiles[*]

Ludovic Courtès Marc-Olivier Killijian David Powell Matthieu Roy

prénom.nom@laas.fr
LAAS-CNRS
7 Avenue du Colonel Roche
31077 Toulouse cedex 4
France

RÉSUMÉ

Nous présentons les fonctionnalités d'un service de sauvegarde coopérative pour dispositifs mobiles. Ce service repose sur la collaboration entre dispositifs pour assurer la sauvegarde et le recouvrement des données de chaque dispositif. Nous identifions les propriétés de sûreté de fonctionnement que l'on est en mesure d'attendre d'un tel service. Nous analysons les systèmes de sauvegarde coopérative pair-à-pair décrits dans la littérature afin d'identifier d'éventuelles fonctionnalités transposables à l'environnement mobile. Enfin, nous concluons sur les spécificités de cet environnement et identifions les axes de recherche à explorer.

Mots-clef : Sauvegarde, dispositifs mobiles, coopération, pair-à-pair.

SUMMARY

We present the features of a collaborative backup service for mobile devices. This service relies on collaboration among peers in order to provide data backup and recovery. We identify the expected dependability properties for such a service. We then survey peer-to-peer collaborative backup systems described in the literature and identify mechanisms relevant to the mobile environment. We conclude on the specificities of this environment and identify future research directions.

Keywords : Mobile applications, data back-up, collaboration, peer-to-peer.

1. INTRODUCTION

Nous abordons la problématique liée à la conception d'un intergiciel fournissant des mécanismes de sûreté de fonctionnement à des dispositifs mobiles dotés de moyens de communication sans fil. Nous considérons des dispositifs très dynamiques et mobiles, n'ayant accès à une infrastructure fixe de communication (un réseau local ou Internet) que par intermittence. Ces dispositifs mobiles doivent être capables de communiquer entre eux, lorsqu'ils sont à proximité physique, en utilisant des moyens de communication *ad hoc* à un saut ou à plusieurs sauts. Cependant, nous souhaitons que le service développé soit utile à une large palette de systèmes, allant aussi bien de dispositifs très mobiles n'ayant que rarement accès à Internet, à l'autre extrême représenté par des machines connectées en permanence à Internet et peu ou pas mobiles. Il est important de noter que dans ce schéma, nous faisons l'hypothèse que les participants à ce service n'ont aucune relation de confiance au préalable.

Le service a pour objectif de permettre aux dispositifs sur lesquels il s'exécute de tolérer les fautes pouvant entraîner la perte de données : perte ou vol du dispositif mobile, effacement accidentel de données par l'utilisateur. Pour résister à de telles fautes, il doit fournir les moyens de sauvegarder les données de l'utilisateur sur un périphérique tiers, en utilisant les moyens de communication dont il dispose.

À l'heure actuelle, les utilisateurs de systèmes mobiles, qu'il s'agisse d'ordinateurs portables (*laptop*) ou d'assistants numériques personnels (PDA), effectuent le plus souvent la sauvegarde de leurs données lorsqu'ils ont accès à leur machine de bureau en synchronisant les données entre les deux machines. Typiquement, les premières générations d'assistants personnels avaient pour seul moyen de communication un système de courte portée, généralement un câble série ou un port infrarouge, demandant à l'utilisateur de se trouver à proximité de la machine sur laquelle s'effectue la sauvegarde. Les dispositifs mobiles actuels disposent généralement de plusieurs interfaces de communication (par exemple IEEE 802.11, Bluetooth). Lorsqu'une infrastructure réseau est disponible dans leur environnement, par exemple des points d'accès Wifi, ces dispositifs mobiles peuvent avoir l'occasion de se connecter à leur machine de bureau pour y sauvegarder leurs données.

Dans la pratique, il est pourtant rare que cette possibilité soit utilisée pour effectuer à distance des sauvegardes, et ce pour plusieurs raisons :

- elle nécessite que la machine distante soit en fonctionnement, connectée à Internet et accessible ;
- l'accès à une infrastructure par des moyens de communication sans fil est encore rare (et cher) et il peut s'écouler

[*] Ces travaux s'intègrent dans le cadre du projet MoSAIC (*Mobile System Availability, Integrity and Confidentiality, http://www.laas.fr/mosaic/*) partiellement financé par l'Action Concertée Incitative (ACI) Sécurité & Informatique du Ministère délégué à la Recherche et aux Nouvelles Technologies.

longtemps avant que le système mobile soit en mesure de se connecter à Internet ;

- enfin, à notre connaissance, les logiciels capables de tenter automatiquement une sauvegarde sur la machine de bureau sont encore rares.

Une autre solution consiste à utiliser les services d'un tiers de confiance qui garantit la disponibilité de ses serveurs de sauvegarde. Différentes offres commerciales permettent de sauvegarder ses données moyennant un abonnement annuel pour des capacités de stockage souvent limitées.

En revanche, nous pensons que l'avènement de dispositifs mobiles équipés de moyens de communication sans fil de courte portée offre des possibilités d'interactions *entre pairs* dont pourrait profiter un système de sauvegarde coopératif. En effet, l'utilisation répandue de systèmes mobiles communicant va permettre des interactions fréquentes mais de courte durée. Nous souhaitons tirer profit de ces interactions : à chaque rencontre de deux systèmes mobiles, le service va automatiquement initier une demande de sauvegarde pour une partie de ses données. En contrepartie, il devra rendre le même service à la communauté. Pour être pratique, ces transactions devront s'effectuer *automatiquement*, sans aucune intervention de l'utilisateur. De cette façon, chaque utilisateur pourra sauvegarder tout ou partie de ses données, et ce avec une granularité assez fine compte-tenu de la fréquence des rencontres que l'on peut envisager.

Nous présentons dans la section suivante les objectifs que nous souhaitons atteindre, notamment en termes de sûreté de fonctionnement, en fonction des problèmes nouveaux qu'il pose. La section 3 présente les travaux décrits dans la littérature en matière de réseaux pair-à-pair, de stockage réparti, et de sauvegarde coopérative dont pourra s'inspirer notre système de sauvegarde. Enfin, la section 4 présente nos conclusions quant à l'apport des systèmes de sauvegarde pair-à-pair par rapport à nos objectifs ainsi que nos pistes de recherche à venir.

2. OBJECTIFS ET PROBLÉMATIQUE

Un mécanisme de sauvegarde est une fonctionnalité *a priori* indépendante des applications et qui s'inscrit dans une problématique orthogonale à celles-ci. Dans cette section, nous décrivons les objectifs du service de sauvegarde coopérative pour dispositifs mobiles que nous baptisons *ColBack* (*Collaborative Backup*). Ses objectifs peuvent eux-mêmes être décomposés en un ensemble de fonctionnalités (sous-section 2.1) et un ensemble de caractéristiques non fonctionnelles que l'on peut attendre d'un tel service, telles que des propriétés de sûreté de fonctionnement (sous-section 2.2).

Nous utiliserons par la suite la terminologie suivante : un *propriétaire de données* est un dispositif mobile tenant le rôle de client du service de sauvegarde coopérative ; les *contributeurs* sont les dispositifs participant à ce service, c'est-à-dire les systèmes stockant des données pour des propriétaires. D'une manière générale, un *participant* est un dispositif mobile qui participe au service (contributeur) ou en profite (propriétaire).

2.1. Fonctions du service

Nous présentons ici les principales fonctionnalités de ColBack : la découverte et l'allocation de ressources effectuées lors de la création de copies, puis le recouvrement des données lors de la restauration.

2.1.1. Découverte et allocation de ressources

La première nécessité pour ColBack est de *découvrir* l'espace de stockage à disposition dans son environnement, puis d'obtenir l'*allocation* d'espace sur le support de stockage d'un tiers pour y stocker tout ou partie des données à sauvegarder. Compte-tenu des scénarios envisagés en termes de connectivité du dispositif, allant d'un accès permanent à une infrastructure jusqu'à une communication uniquement en mode *ad hoc*, plusieurs critères vont influencer le choix de la méthode de découverte et d'allocation de ressources :

- le *coût de la communication*, qui correspond à la quantité d'énergie requise et au nombre de sauts nécessaires pour atteindre un nœud dans le cas d'un réseau *ad hoc* ; lorsqu'un accès direct à Internet est disponible, ce coût est indépendant de la distance séparant deux nœuds ; de même la quantité d'énergie requise pour la communication n'est pas un critère déterminant pour des dispositifs branchés sur le secteur ;
- la *densité* de l'environnement dans lequel évolue le dispositif mobile, c'est-à-dire le nombre de nœuds avec lesquels peut communiquer un nœud influence la probabilité de trouver de l'espace de stockage à proximité ;
- la *volatilité* des connexions (déconnexions fréquentes et parfois permanentes dans le cas d'un réseau *ad hoc*, ou au contraire connexions stables et quasi permanentes entre deux nœuds communiquant directement *via* une infrastructure) influence la granularité des fragments à sauvegarder.

Ces critères permettront de définir différentes stratégies de découverte et d'allocation de ressources suivant les scénarios considérés.

2.1.2. Recouvrement des données

Comme nous l'avons vu en introduction, chaque propriétaire sauvegarde ses données par fragments, au gré des rencontres qu'il fait avec des contributeurs. De ce fait, la principale difficulté de ColBack est de pouvoir recouvrer les données sauvegardées. Nous faisons l'hypothèse que, dans la plupart des scénarios, chaque participant aura accès de manière intermittente à Internet et qu'il pourra attendre ces moments pour récupérer ses propres données si nécessaire (rôle du propriétaire), ou en profiter pour mettre à disposition les données qu'il a lui-même stockées (rôle du contributeur). Une autre possibilité est que les participants n'aient jamais accès à Internet et n'établissent des connexions entre eux qu'au travers d'un réseau *ad hoc*. Dans [15], nous envisageons deux approches au recouvrement des données :

- l'approche *pro-active* (ou *push*), où le contributeur, dès qu'il a accès à Internet, envoie les données à leur propriétaire dans une « boîte aux lettres » prédéterminée ;
- l'approche *réactive* (ou *pull*), où le propriétaire va interroger le réseau de participants ; cette recherche pourrait se faire par mots clef ou méta données tels que « l'ensemble des données que j'ai sauvegardées jeudi matin ».

L'approche réactive paraît adaptée surtout à des réseaux de petite échelle. Nous reviendrons dans la section 3 sur les solutions permettant de mettre en œuvre la boîte aux lettres dont il est question dans la première approche.

2.2. Sûreté de fonctionnement du service

ColBack a pour but d'améliorer la sûreté de fonctionnement des services et données supportés par des dispositifs mobiles. Dans cette section, nous cherchons donc à définir les propriétés de sûreté de fonctionnement que l'on peut attendre de ce service.

2.2.1. Intégrité et cohérence des données sauvegardées

Le service de sauvegarde doit fournir la garantie que les données restaurées par un participant sont cohérentes et intègres. Toute corruption de données sauvegardées, qu'elle soit intentionnelle ou non (par exemple due à une faute logicielle ou matérielle sur la machine stockant les données), doit pouvoir être détectée par son propriétaire lors de la restauration.

Les protocoles réseau et les supports de stockage utilisent largement des codes détecteurs d'erreur ou correcteurs d'erreur pour tolérer les fautes matérielles ou logicielles. Pour résister aux corruptions intentionnelles, il est cependant nécessaire de garantir également l'authenticité des données : le ou les propriétaires des données doivent être sûrs qu'il s'agit bien des données qu'ils ont sauvegardées. Nous présenterons en section 3.3.1 des techniques permettant d'atteindre cet objectif.

2.2.2. Confidentialité des données

Comme nous l'avons vu, les participants au service de sauvegarde coopérative sont amenés à stocker leurs données sur les machines d'autres participants en lesquels ils n'ont aucune relation de confiance *a priori*. Les informations sauvegardées peuvent être sensibles et, en tant que telles, ne doivent pas pouvoir être exploitées par celui qui les stocke. La technique de fragmentation, redondance, dissémination ou FRD [10] permet précisément de garantir la confidentialité des données. Les données sont découpées en fragments puis dispersées entre différents sites de stockage de telle sorte qu'aucun site de stockage n'ait d'information sur l'origine des fragments qu'il stocke. Les données, ou leurs fragments, peuvent être chiffrés pour en améliorer la confidentialité. Bien entendu, cette fragmentation complique le recouvrement des données sauvegardées (section 2.1.2). Nous reviendrons en section 3 sur la mise en œuvre de telles techniques.

La technique de FRD a en outre l'avantage de convenir à nos hypothèses de départ : compte-tenu des interactions éphémères entre systèmes mobiles, les données doivent nécessairement être fragmentées ; de plus, la mobilité entraînera de fait une dissémination des données.

2.2.3. Disponibilité des données

Il y a deux manières d'aborder la question de la disponibilité des données sauvegardées. Du point de vue du contributeur se pose la question du choix des données qu'il devra tôt ou tard effacer pour libérer son espace de stockage. Du point de vue du propriétaire, il s'agira d'avoir une assurance dans sa capacité à restaurer ses données ultérieurement. Enfin, la disponibilité des données ne doit pas être mise en danger par des malveillances tel que des attaques en déni de service.

Choix des données à effacer. Malgré l'utilisation d'algorithmes visant à optimiser l'utilisation de l'espace de stockage, il sera très vite nécessaire pour chaque participant d'effacer certaines données. Le mécanisme qui permettra de décider des données à effacer n'est cependant pas évident compte tenu de la contradiction entre les deux hypothèses suivantes :

1. celui le plus à même de savoir quelle version sauvegardée peut être effacée est le propriétaire des données lui-même ;
2. les participants peuvent être déconnectés et il n'existe *a priori* aucune relation de confiance entre eux.

La première hypothèse nous donne à penser que le propriétaire des données sauvegardées devrait choisir lui-même la version de ses données à effacer : il peut s'agir simplement de la plus vieille sauvegarde, ou bien d'une version plus récente mais moins importante à ses yeux parce qu'elle ne représente pas une étape importante de son travail [27]. Cependant, d'après la deuxième hypothèse formulée, un contributeur (i) ne peut pas compter sur le propriétaire des données pour le prévenir de leur péremption et (ii) ne souhaitera pas dépenser d'énergie pour essayer de reprendre contact avec les propriétaires des données qu'il stocke.

Par conséquent, les participants *doivent* prévoir la possibilité d'une prise de décision unilatérale d'effacer des données si, par exemple, l'utilisateur des ressources n'a pas pris contact avec le contributeur *depuis un certain temps*. Une sémantique précise des obligations mutuelles devra donc être définie et nous en verrons des exemples dans la section 3.3.3.

Duplication. La fréquence à laquelle un propriétaire souhaitant restaurer ses données pourra entrer en relation avec ses contributeurs a un impact direct sur la disponibilité de ses données. Pour résoudre ce problème, une solution évidente est de faire des sauvegardes redondantes, sur des participants indépendants[1], permettant ainsi de récupérer ses données malgré la défaillance d'un ou plusieurs de ses contributeurs.

Résistance au déni de service. Plusieurs types d'attaques par déni de service peuvent être envisagés sur le service de sauvegarde coopérative :

- attaque par *égoïsme*, où un participant profite du service tout en refusant d'y contribuer ;
- attaque par *inondation*, où un propriétaire inonde de requêtes de sauvegarde les autres participants, empêchant de fait les autres propriétaires de bénéficier du service ;
- attaque par *rétention de données*, où un contributeur refuse de rendre les données qu'il stocke, intentionnellement ou pas (par exemple suite à une défaillance).

Nous reviendrons dans la section 3 sur des mécanismes utilisés dans les systèmes pair-à-pair qui peuvent être mis en œuvre pour tolérer ce genre de comportement.

3. SYSTÈMES DE SAUVEGARDE PAIR-À-PAIR

Dans cette section, nous étudions dans quelle mesure l'état de l'art en systèmes de sauvegarde pair-à-pair fournirait des

[1]Notons qu'il est par ailleurs nécessaire d'optimiser le volume global des données sauvegardées. Nous y reviendrons dans la section 3.2.2.

mécanismes aptes à faciliter la mise en œuvre d'un service de sauvegarde coopérative de données pour dispositifs mobiles. Nous donnons, dans un premier temps, une présentation rapide des différents systèmes de sauvegarde pair-à-pair étudiés, puis nous évoquons leur apport dans différents domaines par rapport à nos objectifs.

3.1. Systèmes étudiés

Plusieurs travaux récents traitent de la sauvegarde coopérative. Ils s'inspirent des nombreux travaux portant sur le stockage réparti de fichiers [9,17,25] et le partage de fichiers [2,4]. Tous s'intéressent à la sauvegarde coopérative pour des stations fixes, ayant un accès très régulier à Internet. Il n'existe à notre connaissance, aucun projet qui prenne en compte des systèmes mobiles ayant une connexion intermittente à une infrastructure.

Les premiers travaux décrivant un système de sauvegarde entre pairs sont ceux de Elnikety et al. [11]. Par rapport aux fonctions d'un service de sauvegarde (localisation des ressources, création de copies des données, restauration des données), ce système est assez simple. Un serveur central est utilisé pour trouver des partenaires. Aucun effort n'est fait pour ne sauvegarder que des incréments afin de réduire le temps nécessaire pour effectuer la sauvegarde; l'intégralité des données est envoyée aux partenaires de sauvegardes. Les auteurs décrivent bon nombre d'attaques possibles dont certaines ont été présentées en section 2.2. Nous reviendrons sur les autres attaques, plus spécifiques, par la suite.

Le système *Pastiche* [8] et son extension *Samsara* [7], sont beaucoup plus exhaustifs. Les mécanismes de découverte de ressources, de stockage, et de localisation des données qui sont proposés sont totalement décentralisés et autogérés. Chaque nouvel entrant choisi un ensemble de *partenaires* en prenant en compte différents critères dont la latence des communications, et traite ensuite directement avec eux. Des mécanismes pour minimiser la quantité de données à échanger lors de sauvegardes successives sont également proposés. Enfin, Samsara met en œuvre une solution qui résout les problèmes de juste contribution au service et de résistance aux attaques par déni de service.

D'autres travaux cherchent à résoudre certaines limitations de Pastiche et Samsara, ou proposent de méthodes alternatives, moins complexes. C'est le cas de *Venti-DHash* [29], inspiré par le système d'archivage *Venti* [24] du système d'exploitation Plan 9. Il propose un stockage totalement réparti entre tous les participants, au lieu de sélectionner une fois pour toute un ensemble de partenaires comme le propose Pastiche. Une approche hybride à la localisation et au stockage de données est apportée par *PeerStore* [19] où chaque participant traite en priorité avec un ensemble de partenaires sélectionnés au départ (comme dans Pastiche) mais est capable de ne sauvegarder que les données n'ayant pas encore été sauvegardées. Enfin, *pStore* [1] ainsi que ABS [5] s'inspirent des systèmes de gestion des révisions pour proposer, entre autres, une meilleure utilisation des ressources.

Le système de sauvegarde coopérative *FlashBack* [20] est particulièrement proche de nos préoccupations puisqu'il cible la sauvegarde au sein d'un réseau personnel (PAN). Cependant, de par les caractéristiques d'un tel réseau, ses objectifs diffèrent sensiblement des nôtres. Au sein d'un PAN, tous les dispositifs se font confiance puisqu'ils appartiennent à une même personne. En outre, les auteurs admettent que l'ensemble des dispositifs constitutifs d'un PAN est appelé à rester relativement constant dans le temps puisqu'il s'agit des dispositifs qu'un utilisateur transporte avec lui. Ces deux hypothèses sont en contradiction avec celles que nous faisons : d'une part, aucune relation de confiance préalable n'est requise entre les participants et d'autre part, ceux-ci sont mobiles. Pour ces raisons, nous nous concentrons essentiellement sur les systèmes de sauvegarde pair-à-pair évoqués précédemment.

Par la suite, nous évoquerons les architectures adoptées par les différents projets, puis nous présenterons les techniques qu'ils mettent en œuvre pour atteindre certains des objectifs évoqués au chapitre précédent.

3.2. Aspects fonctionnels

Nous abordons successivement deux aspects spécifiques à ces systèmes de sauvegarde pair-à-pair, à savoir : les mécanismes de découverte et d'allocation de ressources, et les techniques visant à réduire une duplication inutile des données.

3.2.1. Découverte et allocation de ressources

Parmi les systèmes étudiés, on retrouve deux principales approches à la répartition des blocs de données à sauvegarder :

- stockage réparti au sein de groupes particuliers de participants ou *partenaires*;
- stockage réparti entre tous les participants au moyen d'une *table de hachage répartie* (ou DHT) qui a pour propriété de répartir de manière homogène toutes les données sauvegardées.

Dans le premier cas, les relations entre partenaires sont assez simples : chaque participant choisit, à son entrée, un ensemble de partenaires puis leur envoie directement, à chaque sauvegarde, ses données. Le système proposé par Elnikety et al. se contente d'envoyer à intervalle régulier l'ensemble des données à sauvegarder. Les autres systèmes, quant à eux, choisissent les données à transmettre en fonction des versions précédemment sauvegardées. Dans Pastiche, chaque participant choisi également un ensemble de partenaires qui ne changera pas ou peu par la suite. Enfin, les dispositifs participant à FlashBack, au sein d'un PAN, choisissent en priorité pour partenaires les dispositifs qui sont le plus souvent à proximité.

La deuxième approche repose sur une technique fondamentale des réseaux de partage des données pair-à-pair, les « *réseaux virtuels* » ou *overlay networks*, abondamment décrits dans la littérature [26]. Une DHT [25] est un mécanisme de stockage réparti de blocs de données basé sur un réseau virtuel. Chaque nœud du réseau est responsable des blocs dont l'identifiant est proche (numériquement) de son identifiant. Les blocs sont donc répartis de manière homogène dans la DHT si leurs identifiants sont bien répartis. Venti-DHash et pStore suivent cette approche.

L'utilisation d'une DHT pour stocker les données sauvegardées a deux inconvénients :

- le coût de la migration des données lors de l'entrée ou du départ d'un participant peut être élevé en termes d'utilisation de la bande passante [19];
- une DHT répartit automatiquement les données de manière homogène entre tous les participants, indépendamment de l'espace de stockage qu'il consomme; par conséquent, utiliser une DHT empêche d'assurer la justesse des contributions de chacun.

Pour ces raisons, PeerStore propose une approche hybride où les données sont échangées directement entre partenaires, tandis que les méta-informations relatives aux blocs (en l'occurrence, les associations entre les identifiants de bloc et la liste des participants qui en stockent un exemplaire) sont stockées dans une DHT. Comme nous le verrons par la suite, cela procure en outre une plus grande flexibilité à PeerStore.

3.2.2. *Réduction de la duplication inutile de données*

Alors qu'un certain niveau de redondance des données sauvegardées est nécessaire pour tolérer diverses fautes comme nous l'avons vu en section 2.2.3, il est aussi nécessaire de ne pas dupliquer inutilement des données entre partenaires ou participants (dans l'espace) ni entre versions successives d'un même fichier (dans le temps). Les différents systèmes étudiés utilisent des mécanismes permettant de contrôler le nombre d'instances d'un même bloc de données. On parle de *support de stockage à instance unique* ou *stockage convergent*. Cette propriété est obtenue en indexant les données en fonction de leur contenu [24].

Cette approche peut s'avérer intéressante lorsque les participants ont beaucoup de données en commun. Dans Pastiche [8], par exemple, les auteurs font l'hypothèse que les participants vont vouloir sauvegarder toutes les données présentes sur leur machine. Parmi ces données, une bonne partie du système d'exploitation et des applications a des chances d'être commune à de nombreux participants ; des participants collaborant sur un même projet ont également beaucoup de données en commun. Dans cette optique, le stockage à instance unique a donc un apport considérable.

Toutefois, dans le cadre du système de sauvegarde coopérative que nous envisageons, seules les données critiques des utilisateurs, le plus souvent des données personnelles, sont à sauvegarder. On peut donc penser qu'il y aura peu ou pas de duplication inutile des données entre dispositifs mobiles participant. Le phénomène de duplication inutile dans le temps peut néanmoins être observé également dans les scénarios que nous considérons, par exemple lorsqu'un propriétaire est amené à sauvegarder plusieurs versions successives d'un même fichier auprès d'un même contributeur.

3.3. Sûreté de fonctionnement

Nous abordons ici les principales techniques décrites dans la littérature pour garantir les propriétés de sûreté de fonctionnement que nous avons identifiées en section 2.2.

3.3.1. *Intégrité et cohérence*

Ces propriétés de sûreté de fonctionnement sont prises en charge par l'*encodage* des fichiers à sauvegarder, c'est-à-dire par le choix des structures de données représentant un fichier. Chacun des systèmes présentés, mis à part celui d'Elnikety et al., fragmente systématiquement les fichiers à sauvegarder. Cela est nécessaire pour réduire la duplication inutile des données dans l'espace et dans le temps. C'est aussi nécessaire pour assurer une répartition homogène des données sur une DHT, et permettre l'établissement de contributions justes comme nous le verrons par la suite.

pStore utilise des structures de données simples pour représenter les fichiers sauvegardés. Les fichiers sont fragmentés en blocs dont la taille peut varier. Dans pStore, les participants sauvegardent en plus des blocs eux-mêmes une liste des blocs

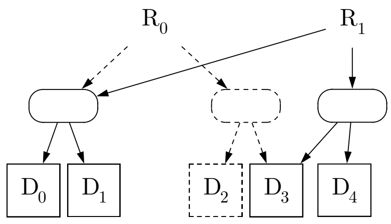

Figure 1. Encodage de plusieurs versions d'une séquence d'octets (fichier) sous forme d'un arbre [24]. Ici, une i-node est partagée entre les deux versions du fichier (dont les racines sont R_0 et R_1). Le bloc de données D_3 est également partagé.

qui contient, pour chaque version du fichier considérée, la liste des identifiants des blocs qui le constituent.Cependant, chaque liste de blocs est indexée par un condensé du chemin du fichier qu'elle représente concaténé à la clef privée de son propriétaire, créant ainsi des espaces de noms de fichiers propres à chaque utilisateur. En pratique, pour restaurer un fichier, il faut donc connaître son chemin et disposer de la clef privée de son propriétaire. Sans cela, il est impossible d'accéder à la liste des blocs du fichier, et donc à ses blocs. C'est la même technique qui est utilisée par PeerStore et une technique similaire pour Pastiche.

Dans ABS, chaque fragment sauvegardé est accompagné d'un bloc de méta-informations. Ces méta-informations sont celles du fichier dont provient le fragment, ainsi que la position du fragment dans le fichier. La clef sous laquelle est stocké l'ensemble est le condensé du contenu du fragment, ce qui permet de maintenir la propriété d'instance unique de chaque fragment. Les méta-informations sont chiffrées avec la clef publique du propriétaire du fichier et l'ensemble (bloc et méta-informations) est signé avec la clef privée du propriétaire. Cette signature garantit l'intégrité de l'ensemble bloc et méta-informations.

Dans Venti-DHash, le codage des fichiers est assuré par Venti. Comme dans un système de fichiers classique, les fichiers sont représentés sous la forme d'un arbre dont les feuilles sont les blocs de données issus de la fragmentation du fichier (figure 1). Les nœuds intermédiaires sont des nœuds d'indirection qui contiennent des pointeurs vers leurs blocs fils. Ici, tous les blocs sont indexés de la même manière, c'est-à-dire par leur condensé. De plus, tous les blocs ont la même taille et le support de stockage sous-jacent ne connaît pas leur sémantique. Pour restaurer une version d'un fichier, il suffit de connaître l'identifiant de l'i-node racine pour cette version.

Toutes ces techniques garantissent dans une certaine mesure l'*intégrité* des données sauvegardées puisque la manière d'adresser les blocs est dépendante de leur contenu (utilisation d'un condensé). Lorsqu'un bloc est restauré, on peut donc immédiatement vérifier qu'il s'agit bien du bloc demandé. Cependant, seul Pastiche [8] aborde la question de la cohérence des données perçue par l'utilisateur telle que nous l'évoquions en section 2.2.1. Étant en partie implémenté sous la forme d'un système de fichiers, Pastiche a la possibilité de copier les blocs qui seront modifiés pendant le processus de sauvegarde, garantissant ainsi la cohérence de ce qui est sauvegardé.

3.3.2. Confidentialité des données

Concernant la confidentialité des données, la technique utilisée par la plupart des systèmes de sauvegarde cités et également par beaucoup de systèmes de partage de fichiers est celle du *chiffrement convergent*. L'objectif est de disposer d'un méthode de chiffrement qui ne dépende pas de celui qui effectue le chiffrement. Le chiffrement convergent est donc un chiffrement symétrique dont la clef est le condensé du bloc à chiffrer. C'est ensuite ce bloc chiffré qui est stocké chez les partenaires ou autres participants. Pour désigner puis déchiffrer un bloc, il est donc nécessaire de connaître son identifiant, c'est-à-dire le plus souvent le condensé du bloc chiffré, et sa clef, c'est-à-dire le condensé du bloc en clair. Ce couple est parfois appelé *clef-condensé* ou CHK (pour *content hash key* [2]).

3.3.3. Disponibilité

Dans cette section, nous décrivons les techniques décrites dans la littérature pour améliorer la disponibilité des données : la duplication et le choix des données à effacer.

Duplication. Pour les systèmes de sauvegarde coopérative où chaque nœud choisit un ensemble de partenaires (Pastiche, PeerStore), le mécanisme de duplication est relativement simple. Dans Pastiche, chaque nouveau participant recherche 5 autres participants ayant un grand nombre de données en commun avec lui ; ces 5 participants deviennent alors ses partenaires de sauvegarde. Il peut donc tolérer la défaillance de 4 de ces nœuds. Dans PeerStore, le choix des partenaires s'effectue d'une manière différente. Toutefois, les auteurs précisent qu'idéalement chaque participant a autant de partenaires que d'exemplaires de ses données, ce qui nous ramène au même schéma que pour Pastiche.

Pour les systèmes basés sur une DHT, par hypothèse, l'ensemble des données stockées dans la DHT est réparti de manière homogène entre les nœuds. Par conséquent, pour tolérer le départ ou la défaillance de participants, les données stockées *doivent* être dupliquées pour ne pas être perdues. En pratique, les blocs sont généralement dupliqués par le nœud qui en est responsable sur un petit nombre de ses voisins dans l'espace des identifiants, et gardés en cache par les nœuds se trouvant sur le chemin permettant d'y accéder [9,25,26].

Enfin, il est aussi possible d'utiliser des blocs de parité ajoutés aux fichiers à sauvegarder [5] ou des codes protégeant contre la perte de données (*erasure codes*) qui permettent de reconstruire un bloc simplement à partir d'un sous-ensemble des fragments de ce bloc. Venti-DHash utilise cette seconde possibilité en stockant des fragments de ce bloc sur les successeurs du nœud qui en est responsable. La tolérance à la perte de fragments peut être ajustée en trouvant un compromis avec la taille de ces fragments.

ABS, où les données sont stockées dans une DHT, propose une solution alternative : les propriétaires peuvent choisir la clef sous laquelle stocker un bloc de données. En premier lieu, lors de l'insertion d'un bloc dans la DHT, les participants essayent de l'insérer en prenant pour clef son condensé. Si cela échoue, par exemple parce que la machine responsable de cette clef s'est retirée, il est possible de choisir pour nouvelle clef un condensé de la clef précédente (on parle de *rehashing*), déplaçant de ce fait les données sur un autre nœud.

Choix des données à effacer. Pastiche, pStore et ABS donnent la possibilité d'effacer des données sauvegardées. Seuls les propriétaires des données peuvent le faire car les demandes d'effacement doivent être signées avec la clef privée du propriétaire. De plus, à chaque bloc est associée une liste des propriétaires afin qu'un bloc ne soit effacé que lorsque tous ses propriétaires l'ont demandé.

PeerStore, en revanche, ne permet pas d'effacer des données sauvegardées. L'inclusion d'une telle fonctionnalité est compliquée par le fait que tous les blocs sauvegardés par un participant n'ont pas fait l'objet d'un accord avec un partenaire (section 3.2.2).

3.3.4. Résistance aux attaques en déni de service

La mise en œuvre de mécanismes permettant de résister aux attaques par déni de service est un sujet à part entière qui demanderait plus de place pour être traité en détail. Nous n'aborderons donc ici que les principales solutions proposées pour chacune des attaques que nous avons identifiées en section 2.2.3.

Égoïsme et inondation. L'attaque par égoïsme est un problème rencontré par tous les systèmes de partage de ressources, en particulier les systèmes de partage de fichiers. De nombreuses solutions ont été proposées. Nous ne nous intéressons ici qu'à celles dédiées aux systèmes de sauvegarde.

Il faut d'abord remarquer qu'il est pratiquement impossible de garantir l'équité des contributions dans une DHT : par hypothèse, toutes les données sont réparties de manière homogène entre les nœuds, indépendamment des ressources utilisées par le nœud. Par conséquent, les systèmes pStore et Venti-DHash ne sont pas résistants à ce type d'attaque. Ils se placent dans le schéma de la « tragédie des biens communs » décrite par Hardin [13] : l'espace de stockage est un bien commun, son coût d'utilisation est partagé entre tous et l'intérêt de chaque participant est donc de profiter de cette ressource. La technique de *rehashing* d'ABS (cf. section 3.3.3) peut servir à équilibrer la charge sur la DHT mais elle ne prend pas en compte l'utilisation des ressources de chacun et est difficilement contrôlable.

PeerStore propose une solution assez simple : tous les échanges au sein d'un couple de partenaires doivent être *symétriques*, c'est-à-dire que chacun doit offrir la même quantité d'espace que ce qu'il consomme. Pour trouver des partenaires, chaque nouvel entrant diffuse une offre pour une certaine quantité d'espace de stockage et reçoit par les intéressés des propositions incluant une offre qui peut être différente. C'est au demandeur de décider si l'échange lui convient.

Pastiche ne traite pas ce problème mais Samsara apporte des solutions. Les auteurs y proposent l'établissement d'échanges symétriques par l'obtention d'un droit au stockage représenté par une sorte de capacité lorsqu'un contributeur s'engage à stocker des données. Les droits obtenus par un contributeur peuvent être cédés, lorsqu'il prend le rôle de propriétaire, à un autre participant. Enfin, il est possible à chaque participant de vérifier que ses droits au stockage sont respectés. La littérature des systèmes pair-à-pair de partage de fichiers décrit des solutions plus élaborées basées sur les notions de *confiance*, de *réputation*, ou de *micro-économie* [12,18].

Les propositions basées sur des rapports symétriques [7,19] ont l'avantage d'être résistantes aux attaques par inondation, où un nœud cherche à utiliser toutes les ressources du réseau. Au contraire, les DHT ne sont pas résistantes aux attaques par inondation de par la répartition homogène des données qu'elles font.

Rétention de données. Elnikety et al. insistent sur la nécessité de pouvoir tolérer la rétention non intentionnelle, par exemple lorsqu'elle est due à des déconnexions, tout en étant en mesure de « punir » les abus.

Les solutions qu'ils proposent sont d'une part de vérifier que les partenaires stockent bel et bien les données qu'ils sont censés stocker, et d'autre part d'introduire des règles pour la tolérance aux fautes temporaires. La vérification des données sauvegardées se fait par *défis* réguliers, c'est-à-dire par l'envoi de demandes de lecture d'un bloc choisi au hasard au contributeur. La tolérance aux fautes temporaires (déconnexions) se fait par l'établissement d'une *période de grâce* pendant laquelle un participant peut être indisponible sans pour autant que ses données soient effacées. Cependant, ce mécanisme pourrait être utilisé pour profiter des ressources disponibles sans y contribuer. Par conséquent, les auteurs proposent en outre de définir une période d'essai, plus longue que la période de grâce, pendant laquelle les sauvegardes et défis sont autorisés mais pas les restaurations.

Cette technique des défis est reprise par l'ensemble des autres systèmes de sauvegarde coopérative étudiés. En revanche, au lieu de demander la lecture d'un seul bloc par défi, les autres systèmes envoient typiquement une liste de blocs; en réponse, ils reçoivent une simple signature de l'ensemble de ces blocs, ce qui limite l'utilisation de la bande passante [7,19].

En revanche, d'autres systèmes proposent des méthodes de tolérance aux déconnexions et de punition associée. Samsara [7] en particulier propose une punition progressive pour les participants ne répondant pas, plutôt que l'approche « tout ou rien » de la période de grâce. Dans Samsara, lorsqu'un nœud n'arrive pas à joindre un de ses partenaires, il détruit un bloc choisi au hasard. La probabilité d'effacement d'un bloc donné est choisie telle que, compte tenu du nombre de copies des données du nœud déconnecté, la probabilité qu'un bloc ait été effacé de chaque copie ne devient significative qu'après un nombre important de non-réponses auprès de ses partenaires. Là encore, PeerStore utilise la même approche.

4. CONCLUSIONS ET FUTURES DIRECTIONS

Les systèmes de sauvegarde coopérative que nous venons de présenter sont une source d'inspiration importante pour ColBack. Toutefois, certaines solutions proposées par ceux-ci ne correspondent pas à nos objectifs et un certain nombre de questions restent ouvertes pour prendre en compte le fait que les dispositifs mobiles seront peu, voire jamais, connectés à Internet. Nous présentons ici les pistes de recherche envisagées pour chacun de ces problèmes.

Points communs et différences entre pair-à-pair et *ad hoc*. Les réseaux *ad hoc* de dispositifs mobiles peuvent être vus comme une forme de réseau pair-à-pair puisque les dispositifs interagissent d'égal à égal, de manière décentralisée. Cependant, ce parallèle a ses limites dont certaines sont décrites dans la littérature [16]. La qualité et la bande passante des connexions sont évidemment moins bonnes dans un réseau *ad hoc* qu'elles ne peuvent l'être sur Internet. Les chemins à plusieurs sauts connectant deux dispositifs sont particulièrement instables. De plus, l'ensemble des dispositifs présents dans l'entourage d'un dispositif est en constante évolution. Beaucoup de réseaux pair-à-pair font au contraire l'hypothèse de connexions relativement stables et d'un taux de renouvellement des participants assez faible. Cela est particulièrement le cas pour les réseaux virtuels dits « structurés » tels que ceux sur lesquels reposent les DHT[2]. Par ailleurs, les réseaux virtuels font généralement l'hypothèse d'un mécanisme de désignation fixe (IP ou autre) qui n'est pas forcément disponible sur un réseau *ad hoc*.

La consommation énergétique est une préoccupation cruciale dans le cadre de systèmes mobiles, alors qu'elle est ignorée des travaux portant sur les systèmes pair-à-pair classiques. Il nous faudra tenir compte de cet aspect dans la conception de ColBack et de ses protocoles, en nous inspirant d'autres systèmes ayant des buts et des contraintes similaires [20,21].

Découverte et allocation de ressources en mode *ad hoc*. La découverte de ressources sur les réseaux pair-à-pair diffère également grandement de ce qui peut se faire sur un réseau *ad hoc*. Pour participer à un réseau virtuel, il est généralement nécessaire et suffisant de connaître un participant, par exemple par son adresse IP, laquelle peut être obtenue auprès d'un serveur central de listes de participants, ou encore par diffusion de requêtes de découverte dans un réseau local [26]. Dans un réseau *ad hoc*, la découverte de dispositifs mobiles dans son environnement physique est fournie par le protocole réseau [16]. La découverte de ressources est cependant plus complexe du fait de l'instabilité du réseau et fait l'objet de nombreux travaux dont nous pourrons nous inspirer [14].

Bien entendu, à cause du caractère dynamique des réseaux *ad hoc*, les algorithmes d'allocation de ressources des réseaux pair-à-pair (DHT, ou au sein d'un groupe de partenaires) ne sont pas adaptés à un contexte *ad hoc*. Pour la sauvegarde et le recouvrement de données en mode *ad hoc*, nous pourrons tirer profit d'un grand nombre de travaux relatifs à la dissémination de données au sein d'une communauté de systèmes mobiles communiquants [3,21,22].

Adaptation aux moyens de communication disponibles. Dans la majorité des scénarios que nous considérons, les dispositifs participant auront un accès intermittent à Internet, ou seront de temps en temps connectés à une station elle-même connectée. Ces instants devront donc être mis à profit pour améliorer la qualité du service de sauvegarde. Un scénario possible est que le système mobile envoie dans la « boîte aux lettres » de leur propriétaire (cf. section 2.1.2) les données qu'il a sauvegardées. Dans cette situation, les travaux présentés sur la sauvegarde et le stockage répartis pair-à-pair pourraient constituer une source d'inspiration importante dans la mise en œuvre du mécanisme de boîte aux lettres. Cependant, le système mobile lui-même ne pourrait pas contribuer à ce service de stockage réparti compte-tenu du fait qu'il ne sera que rarement connecté à Internet.

Modèle de confiance. Les modèles d'échange présentés dans la section précédente (mécanisme d'offre et de demande, échanges symétriques [19], établissement de liens symétriques [7], mécanisme de défis) font tous l'hypothèse que les participants sont le plus souvent connectés entre eux. Pour les scénarios que nous envisageons (forte mobilité, renouvellement constant du voisinage des dispositifs, établissement de connexions éphémères entre participants), ces mécanismes ne sont pas applicables. Des solutions pour dispositifs mobiles, inspirées des modèles utilisés dans les réseaux pair-à-pair, existent : dans [28], les auteurs proposent que des informations de réputation soient échangées entre participants, au gré des rencontres. Dans [15], nous proposons que chacun porte sa propre information de réputation, et que celle-

[2] Il existe aussi des réseaux virtuels pair-à-pair dits « non structurés », c'est-à-dire ne reposant pas sur l'utilisation d'algorithmes de routage déterministes [4].

ci puisse être vérifiée par les autres participants. La définition d'un tel mécanisme de réputation auto-portée est encore un problème ouvert.

L'utilisation du mécanisme de boîte aux lettres sur Internet engendre des défis supplémentaires quant à la conception d'un modèle de confiance. Lorsqu'un dispositif propriétaire recouvre ses données *via* Internet, il doit ainsi être capable de récompenser (i) les machines de bureau ayant contribué au service de boîte aux lettres sur Internet et (ii) le dispositif mobile contributeur qui a porté ses données jusqu'à Internet. Il s'agit là aussi d'un problème ouvert.

Architecture. En termes d'architecture logicielle, l'intégration du service de sauvegarde au système d'exploitation nous semble être une voie à explorer. La sauvegarde et l'archivage ont beaucoup en commun avec la gestion des versions. La littérature en matière de systèmes de fichiers gérant les versions [6,8,23,27] a montré combien cette approche offrait d'une part un meilleur contrôle sur les données (par exemple la possibilité de faire de l'« archivage exhaustif » des versions), et d'autre part des opportunités pour optimiser la sauvegarde de versions successives d'un fichier, en termes d'espace disque consommé et de performance.

BIBLIOGRAPHIE

[1] C. Batten, K. Barr, A. Saraf, S. Treptin. pStore : A secure peer-to-peer backup system. MIT-LCS-TM-632, MIT Laboratory for Computer Science, December 2001.

[2] K. Bennett, C. Grothoff, T. Horozov, J. T. Lindgren. An Encoding for Censorship-Resistant Sharing. 2003.

[3] M. Boulkenafed, V. Issarny. AdHocFS : Sharing Files in WLANs. *Proc. of the 2nd Int. Symp. on Network Computing and Applications*, 2003.

[4] Y. Chawathe, S. Ratnasamy, L. Breslau, S. Shenker. Making Gnutella-like P2P Systems Scalable. *Proc. of ACM SIGCOMM 2003*, pages 407–418, 2003.

[5] J. Cooley, C. Taylor, A. Peacock. ABS : The Apportioned Backup System. MIT Laboratory for Computer Science, 2004.

[6] B. Cornell, P. Dinda, F. Bustamante. Wayback : A User-level Versioning File System for Linux. *Proc. of the USENIX Annual Technical Conference, FREENIX Track*, pages 19–28, 2004.

[7] L. P. Cox, B. D. Noble. Samsara : Honor Among Thieves in Peer-to-Peer Storage. *Proc. 19th ACM SOSP*, pages 120–132, 2003.

[8] L. P. Cox, B. D. Noble. Pastiche : Making backup cheap and easy. *5th USENIX OSDI*, pages 285–298, 2002.

[9] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, I. Stoica. Wide-area cooperative storage with CFS. *Proc. 18th ACM SOSP*, pages 202–215, 2001.

[10] Y. Deswarte, L. Blain, J-C. Fabre. Intrusion Tolerance in Distributed Computing Systems. *Proc. of the IEEE Symp. on Research in Security and Privacy*, pages 110–121, 1991.

[11] S. Elnikety, M. Lillibridge, M. Burrows. Peer-to-peer Cooperative Backup System. *Proc. of USENIX FAST 2002*, 2002.

[12] C. Grothoff. An Excess-Based Economic Model for Resource Allocation in Peer-to-Peer Networks. *Wirtschaftsinformatik*, 3-2003, June 2003.

[13] G. Hardin. The Tragedy of the Commons. *Science*, (162), 1968.

[14] A. Helmy. Efficient Resource Discovery in Wireless AdHoc Networks : Contacts Do Help. *Kluwer Academic Publishers*, May 2004.

[15] M-O. Killijian, D. Powell, M. Banâtre, P. Couderc, Y. Roudier. Collaborative Backup for Dependable Mobile Applications. *Proc. of 2nd Int. Workshop on Middleware for Pervasive and Ad-Hoc Computing (Middleware 2004)*, pages 146–149, 2004.

[16] G. Kortuem, J. Schneider, D. Preuitt, T. G. C. Thompson, S. Fickas, Z. Segall. When Peer-to-Peer comes Face-to-Face : Collaborative Peer-to-Peer Computing in Mobile Ad-hoc Networks. *Proc. of P2P*, 2001.

[17] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, W. Weimer, C. Wells, B. Zhao. OceanStore : An Architecture for Global-Scale Persistent Storage. *Proc. of the 9th ASPLOS*, pages 190–201, 2000.

[18] K. Lai, M. Feldman, J. Chuang, I. Stoica. Incentives for Cooperation in Peer-to-Peer Networks. *Workshop on Economics of Peer-to-Peer Systems*, 2003.

[19] M. Landers, H. Zhang, K-L. Tan. PeerStore : Better Performance by Relaxing in Peer-to-Peer Backup. *Proc. of the 4th P2P*, pages 72–79, 2004.

[20] B. T. Loo, A. LaMarca, G. Borriello. Peer-To-Peer Backup for Personal Area Networks. IRS-TR-02-015, UC Berkeley ; Intel Seattle Research (USA), May 2003.

[21] E. B. Nightingale, J. Flinn. Energy-Efficiency and Storage Flexibility in the Blue File System. *Proc. of the 6th OSDI*, 2004.

[22] M. Papadopouli, H. Schulzrinne. Seven Degrees of Separation in Mobile Ad Hoc Networks. *IEEE Conference on Global Communications (GLOBECOM)*, 2000.

[23] Z. Peterson, R. Burns. Ext3cow : The Design, Implementation, and Analysis of Metadata for a Time-Shifting File System. HSSL-2003-03, Hopkins Storage Systems Lab, Department of Computer Science, Johns Hopkins University, USA, 2003.

[24] S. Quinlan, S. Dorward. Venti : A new approach to archival storage. *Proc. of the 1st USENIX FAST*, pages 89–101, 2002.

[25] S. Ratnasamy, P. Francis, M. Handley, R. Karp, S. Shenker. A Scalable Content-Addressable Network. *Proc. of ACM SIGCOMM 2001*, 2001.

[26] A. Rowstron, P. Druschel. Pastry : Scalable, distributed object location and routing for large-scale peer-to-peer systems. *Proc. of the 18th IFIP/ACM Int. Conference on Distributed Systems Platforms (Middleware 2001)*, pages 329–350, 2001.

[27] D. S. Santry, M. J. Feeley, N. C. Hutchinson, A. C. Veitch, R. W. Carton, J. Ofir. Deciding when to forget in the Elephant file system. *Proc. 17th ACM SOSP*, pages 110–123, 1999.

[28] J. Schneider, G. Kortuem, J. Jager, S. Fickas, Z. Segall. Disseminating Trust Information in Wearable Communities. *2nd Int. Symp. on Handheld and Ubiquitous Computing (HUC2K)*, 2000.

[29] E. Sit, J. Cates, R. Cox. A DHT-based Backup System. MIT Laboratory for Computer Science, August 2003.

Storage Tradeoffs in a Collaborative Backup Service for Mobile Devices

Ludovic Courtès Marc-Olivier Killijian David Powell

first-name.last-name@laas.fr
LAAS-CNRS
7 Avenue du Colonel Roche
31077 Toulouse cedex 4
France

ABSTRACT

Mobile devices are increasingly relied on but are used in contexts that put them at risk of physical damage, loss or theft. Yet, due to physical constraints and usage patterns, fault tolerance mechanisms are rarely available for such devices. We consider that this gap can be filled by exploiting spontaneous interactions to implement a collaborative backup service. We define the constraints implied by the mobile environment and analyze how they translate into the storage layer of such a backup system. The design options of the envisaged storage mechanisms are examined based on input from recent work on peer-to-peer systems, distributed file systems, and archival and versioning software. We present our prototype implementation of the storage layer and evaluate the impact of several compression methods. We conclude that whereas some design patterns and techniques used in prior work map well to the requirements of a collaborative backup service for mobile devices, several new issues are raised, e.g., regarding the need for dynamic replica scheduling and dissemination, and tradeoffs between confidentiality, and availability and freshness.

Keywords: Mobile computing, nomadic computing, data back-up, peer-to-peer, fault-tolerance, dependability.

1. INTRODUCTION

Embedded computers are becoming widely available, in various portable devices: personal information systems (PDAs, “smart phones”), digital cameras, portable storage units (e.g., music players), laptops. Most of these devices are now able to communicate using wireless network technologies such as IEEE 802.11, Bluetooth, or Zigbee. Users use such devices to capture more and more data and are becoming increasingly dependent on them. Backing up the data stored on these devices is often done in an *ad hoc* fashion: each protocol and/or application has its own synchronization facilities which one can use when a sister device, usually a desktop computer, is reachable. However, newly created data may be held on the mobile device for a long time before it can be copied. This may be a serious issue since the contexts in which mobile devices are used increase the risks of them being lost, stolen or broken.

Our goal is to leverage the ubiquity of communicating mobile devices to implement a *collaborative* backup service. In such a system, devices participating in the service would be able to use other devices’ storage to back up their own data. Of course, each device would have to contribute some of its own storage resources for others to be able to benefit from the service.

Wired peer-to-peer systems (i.e., on the Internet) paved the way for such services. They showed that excess resources, such as storage, available at the peer hosts could be leveraged to support wide-scale resource sharing. Although the amount of resources available on a mobile device is significantly smaller than that of a desktop machine, we believe that this is not a barrier to the creation of mobile peer-to-peer services. They have also shown that wide-scale services could be created without relying on any infrastructure (other than the Internet itself), in a decentralized, self-administered way. A great benefit of this is that users can participate without having to go through an explicit registration phase. Additionally, from a fault-tolerance viewpoint, peer-to-peer systems provide a high diversity of nodes with independent failure modes [17].

In a mobile context, we believe there are additional reasons to use a collaborative service. For instance, access to a cell phone communication infrastructure (GPRS, UMTS, etc.) may be costly (especially for non-productive data transmission “just” for the sake of backup) while proximity communications are not (using 802.11, Bluetooth, etc.). Similarly, short-distance communication technologies are often more efficient than long-distance ones: they offer a higher throughput (e.g., 54 Mbps with 802.11g) and often require less energy. In some scenarios, infrastructure-based networks are not available but neighboring devices might be accessible using *ad hoc* routing or even single-hop communications.

Our target application-layer service raises a number of interesting issues, in particular relating to trust management, resource accounting and cooperation incentives. While considerable work has been devoted to these topics, in particular with respect to peer-to-peer file sharing systems, we believe that the mobile context raises novel issues due to, for instance, mostly disconnected operation and the consequent difficulty of resorting to centralized or on-line solutions. A preliminary analysis of these issues may be found in [8,18]. In this paper, the focus is on the mechanisms employed at the storage layer of such a service. We investigate the various design options at this layer and discuss potential trade-offs.

In Section 2, we will give an overview of a tentative design for the cooperative backup service and detail the requirements of this service on the underlying storage layer. Section 3 presents several design options for this layer based on the current literature and the particular needs that arise from the kind of devices we target. In Section 4, using a flexible prototype of this storage layer, we will evaluate some storage layer algorithms and discuss the necessary tradeoffs. Finally, we will conclude on our current work and sketch future research directions.

2. COLLABORATIVE BACKUP FOR MOBILE DEVICES

This section gives an overview of the service envisaged and related works. Then we describe the requirements we have identified for the storage layer of the service.

2.1. Design Overview and Related Work

Our goal is to design and implement a collaborative backup system for communicating mobile devices. In this model, mobile devices can play the role of a *contributor*, i.e., a device that offers its storage resources to store data on behalf of other nodes, and a *data owner*, i.e., a mobile device asking a contributor to store some of its data on its behalf. Practically, nodes are expected to contribute as much as they benefit from the system; therefore, they should play both roles at the same time.

For the service to effectively leverage the availability of neighboring communicating devices, the service has to be functional even in the presence of *mutually suspicious device users*. We want users with no prior trust relationships to be able to use the service and to contribute to it harmlessly. This is in contrast with traditional habits where users usually back up their mobile devices' data only on machines they trust, such as their workstation.

This goal also contrasts with previous work on collaborative backup for a personal area network (PAN), such as FlashBack [23], where participating devices are all trustworthy since they belong to the same user. However, censorship-resistant peer-to-peer file sharing systems such as GNUnet [2] have a similar approach to security in the presence of adversaries.

Recently, a large amount of research has gone into the design and implementation of Internet-based peer-to-peer backup systems that do not assume prior trust relationships among participants [1,9,11,20]. There is, however, a significant difference between those Internet-based systems and what we envision: *connectivity*. Although these Internet-based collaborative backup system are designed to tolerate disconnections, they do assume a high-level of connectivity. Disconnections are assumed to be mostly transient, whether they be non-malicious (a peer goes offline for a few days or crashes) or malicious (a peer purposefully disconnects in order to try to benefit from the system without actually contributing to it).

In the context of mobile devices interacting spontaneously, connections are by definition short-lived, unpredictable, and very variable in bandwidth and reliability. Worse than that, a pair of peers may spontaneously encounter and start exchanging data at one point in time and then never meet again.

To tackle this issue, we envision scenarios where each mobile device can *intermittently* access the Internet. The backup software running in those mobile devices is expected to take advantage of such an opportunity by re-establishing contacts with mobile devices encountered earlier. For instance, a contributor may wish to send data stored on behalf of another node to some sort of *repository* associated with the owner of the data. Contributors can thus asynchronously *push* data back to their owners. The repository itself can be implemented in various ways: an email mailbox, an FTP server, a fixed peer-to-peer storage system, etc. Likewise, data owners may sometimes need to query their repository as soon as they can access the Internet in order to *pull* (i.e., restore) their data back.

In the remainder of this paper, we will focus on the design of the storage layer of this service on both the data owner and contributor sides. The next section will detail requirements for the design of that layer that arise from the constraints we have just identified.

2.2. Requirements of the Storage Layer

We now identify several requirements to be fulfilled by the mechanisms employed at the storage layer.

Storage efficiency. Backing up data should be as efficient as possible. Data owners should neither ask contributors to store more data than necessary nor send excessive data over the wireless interface. Failing to do so will waste energy and result in inefficient utilization of the storage resources available in the node's vicinity. The impact of inefficient storage on energy consumption may be important since (i) storage cost will translate into transmission costs and (ii) we can reasonably assume that energy consumption on mobile devices is dominated by wireless communication costs [31]. As a consequence, *compression techniques* will be a key point of the design and implementation of the storage layer on the data owner side.

Small data blocks. Both the occurrence of encounters of a peer within radio range and the lifetime of the resulting connections are unpredictable. Consequently, the backup application running on a data owner's device must be able to conveniently split the data to be backed up into small pieces to ensure that it can actually be transferred to contributors. Ideally, data blocks should be able to fit within the underlying network layer's maximum transmission unit or MTU (2304 octets for IEEE 802.11 [4]); larger payloads get fragmented into several packets, which introduces overhead at the MAC layer, and possibly at the transport layer too.

Backup atomicity. Unpredictability and the potentially short lifetime of connections, compounded with the possible use of differential compression to save storage resources, mean that it is unlikely to be practical to store a set of files, or even one complete file, on a single peer. Indeed, it may even be undesirable to do so in order to protect data confidentiality [10]. Furthermore, it may be the case that files are modified before their previous version has been completely backed up.

The dissemination of data chunks as well as the coexistence of several versions of a file must not have a negative impact on backup consistency as perceived by the end-user: a file should be either retrievable *and* correct or unavailable. This corresponds to the *atomicity* property, one of the four ACID properties[1] commonly referred to in transactional database management systems. The *consistency* property (i.e., the fact that the distributed store, consisting of various contributors, remains in a "legal" state after new data are backed up onto it) is also obviously desirable.

Error detection. Accidental modifications of the data are assumed to be handled by the various lower-level software and hardware components involved, such as the communication protocol stack, the storage devices themselves, the operating system's file system implementation, etc. However, given that data owners are to hand their data to untrusted peers, the storage layer must provide all the mechanisms necessary to ensure that *malicious* modifications to their data are detected with a high probability. Consequently, error-detection codes such as CRCs, which are designed to handle random errors, are not appropriate.

Encryption. Similarly, due to the lack of trust in contributors, data owners may wish to encrypt their data in order to enforce their privacy. While there exist scenarios where there is sufficient trust among the participants such that encryption is not compulsory (e.g., several people in the same working group), encryption is a requirement in the general case.

[1]ACID: atomicity, consistency, isolation, durability.

Beside encryption, data fragmentation and dissemination amongst multiple devices also contribute to improving data confidentiality [10].

Backup redundancy. Redundancy is the *raison d'être* of any data backup system, but when the system is based on cooperation, the backups themselves must be made redundant. First, the cooperative backup software must account for the fact that contributors may crash accidently and lose the data stored on behalf of various data owners. Second, contributor availability is unpredictable in a mobile environment without continuous Internet access. Third, contributors are not fully trusted and may behave maliciously. The literature on Internet-based peer-to-peer backup systems describes a range of attacks against data availability, ranging from data retention (i.e., a contributor purposefully refuses to allow a data owner to retrieve its data) to selfishness (i.e., a participant refuses to spend energy and storage resources storing data on behalf of other nodes) [9,11]. All these uncertainties make redundancy even more critical in a cooperative backup service for mobile devices.

3. DESIGN OPTIONS FOR THE STORAGE LAYER

In this section, we present design options able to satisfy each of the requirements we have identified for the storage layer of our cooperative backup service, namely: storage efficiency, small data blocks, backup atomicity, error detection, encryption and backup redundancy.

3.1. Storage Efficiency

In wired distributed cooperative services, storage efficiency is often addressed by ensuring that a given content is only stored once. This property is known as *single-instance storage* [5]. It can be thought of as a form of compression among several datum units. In a file system, where the "datum unit" is the file, this means that a given content stored under different file names will be stored only once. On Unix-like systems, revision control and backup tools implement this property by using hard links: files unchanged across revisions/snapshots are hard-linked instead of being duplicated [24,29]. Thus, the compression provided by single-instance storage can be considered as compression in space (i.e., across file boundaries and potentially across peer boundaries) and in time (i.e., across subsequent versions of a file).

The single instance property may also be provided at a sub-file granularity, instead of at a whole file level. This allows reduction of unnecessary duplication with a finer-grain. Archival systems [27,38], peer-to-peer file sharing systems [2], peer-to-peer backup systems [9,20], network file systems [26], and remote synchronization tools [34] have been demonstrated to benefit from single-instance storage, either by improving storage efficiency or reducing bandwidth.

Compression based on resemblance detection, that is, *differential compression*, or *delta encoding*, has been extensively studied [16]. Proposals have been made to combine it with other compression techniques such as single-instance storage, even in situations that do not strictly relate to versioning [19,38]. For each file to be stored, an exhaustive search over all stored files is performed to find the most similar file so that only the difference between these two files is stored. However, this technique is unsuitable for our environment since (i) it requires access to all the files already stored, (ii) it is CPU- and memory-intensive, and (iii) the resulting *delta chains* weaken data availability [38].

Traditional lossless compression (i.e., *zip* variants), allows the elimination of duplication *within* single files. As such, it naturally complements inter-file and inter-version compression techniques [38]. Section 4 contains a discussion of the combination of both techniques in the framework of our proposed backup service.

Since files are typically smaller than 8 KiB [12] while lossless compressors usually operate on larger input buffers (for instance, *zlib* uses a 64 KiB buffer for the input data), compressing a bunch of files (e.g., by combining `tar` and `gzip`) rather than individual files usually yields better compression ratios [19]. However, we did not consider this approach suitable for mobile device backup since it may be more efficient to backup only those files (or part of files) that have changed.

There exist a number of application-specific *lossless* compression algorithms, such as those used by the Free Lossless Audio Codec (FLAC) and by the Portable Network Graphics format (PNG). Application-specific differential algorithms have also been developed, e.g., for XML trees [6] and for executable files [14]. There is also a plethora of *lossy* compression algorithms for audio samples, bitmap images, videos, etc.

While using such type-specific algorithms might be beneficial in some cases, in particular, for mobile devices that are capable of digital acquisition (e.g., digital cameras, sound recorders, etc.), we have focused instead on generic lossless compression (e.g., as implemented, by *gzip* and the underlying *zlib* library) which we expect to be valuable with most data types.

3.2. Small Data Blocks

We now consider the options available to (1) chop input streams into small blocks, and (2) create appropriate meta-data describing how those data blocks should be reassembled to produce the original stream.

3.2.1. Chopping Algorithms

As stated in Section 2.2, the size of blocks that are to be handed to contributors of the backup service has to be bounded, and preferably small, in order to match the nature of peer interactions in a mobile environment. There are several ways to cut input streams into bounded blocks. Which algorithm is chosen has an impact on the improvement yielded by single-instance storage.

Splitting input streams into fixed-size blocks is a natural solution. When used in conjunction with a single-instance storage mechanism, it has been shown to improve the amount of unnecessary duplication that can be eliminated across files or across file versions [27]. On the other hand, Manber proposed a content-based stream chopping algorithm [25] that yields better duplication detection across files. The algorithm determines block boundaries by computing Rabin fingerprints on a sliding window of the input streams. This technique is sometimes referred to as *content-defined blocks* [19]. Various applications such as archival systems [38], network file systems [26] and backup systems [9] benefit from this algorithm. Manber's algorithm only allows the specification of an average block size (assuming random input); "pathological" input may yield very small or, conversely, very large blocks. Therefore, lower and upper boundaries on the acceptable size of blocks yielded by this algorithm need to be defined [26]. Section 4 provides a comparison of both algorithms.

3.2.2. Stream Meta-Data

Placement of stream meta-data. Stream meta-data is information that describes which blocks comprise the stream and how they should be reassembled to produce the original stream. Such meta-data can either be embedded along with each data block or stored separately. The main evaluation criteria of a meta-data structure are read efficiency (e.g., algorithmic complexity of stream retrieval, number of accesses needed) and its size (e.g., how the amount of meta-data grows compared to data).

In Hydra [37], the author proposes a solution where each block (actually each share) contains information as to where it belongs: the path of the file it belongs to, the file permission, the location of other shares, etc.

In our opinion, such an approach is inflexible. We suggest instead that stream meta-data (i.e., which blocks comprise a stream) should be separated both from file meta-data (i.e., file name, permissions, etc.) and the file content. This has several advantages:

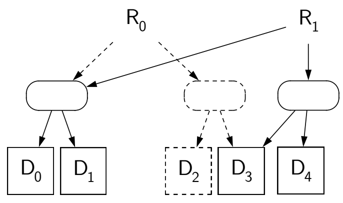

Figure 1. A tree structure for stream meta-data. Leaves represent data blocks while higher blocks are meta-data blocks.

- It allows a data block to be referenced multiple times and hence, allows for single-instance storage at the block level.
- It promotes *separation of concerns*. For instance, file-level meta-data (e.g., file path, modification time, permissions) may change without having to modify the underlying data blocks, which is important in scenarios where propagating such updates would be next to impossible. Separating meta-data and data also leaves the possibility of applying the same "filters" (e.g., compression, encryption), or to use similar redundancy techniques for both data and meta-data blocks. This will be illustrated in Section 4.

It is worth noting that this principle has been followed by on-disk file systems. This made it easy, for instance, to augment regular file systems with versioning facilities [30].

Indexing individual blocks. The separation of data and meta-data means that there must be a way for meta-data blocks to refer to data blocks: data blocks must be indexed or *named*[1]. The block naming scheme must fulfill several requirements. First, it must not be based on non-backed-up user state which would be lost during a crash. Most importantly, the block naming scheme must guarantee that *name clashes* among the blocks of a data owner cannot occur. In particular, block IDs must remain valid in time so that a given block ID is not wrongfully re-used when a device restarts the backup software after a crash. Given that data blocks will be disseminated among several peers and will ultimately migrate to their owner's repository, blocks IDs should remain valid in space, that is, they should be independent of contributor names. This property also allows for *pre-computation* of block IDs and meta-data blocks: stream chopping and indexing do not need to be done upon a contributor encounter, but can be performed *a priori*, once for all. As a result, it saves CPU time and energy, and allows data owners to immediately take advantage of a backup opportunity. A practical naming scheme widely used in the literature will be discussed in Section 3.4.

Indexing sequences of blocks. Byte streams (file contents) can be thought of as sequences of blocks. Meta-data describing the list of blocks comprising a byte stream need to be produced and stored. In their simplest form, such meta-data are a vector of block IDs, or in other words, a byte stream. This means that this byte stream can in turn be indexed, recursively, until a meta-data byte stream is produced that fits the block size constraints. This approach yields the meta-data structure shown in Figure 1 which is comparable to that used by Venti and GNUnet [2,27]. The data structure is a tree whose leaves are data blocks (i.e., fragments of the input stream) and whose intermediate blocks are meta-data blocks containing block IDs. R_0 and R_1 are the "root blocks" of two successive versions of an input stream; as such, some of the blocks they point to are shared. Again, this is similar to the data structures used by on-disk file systems.

Contributor interface. With such a design, contributors need not have knowledge about the actual implementation of block and stream indexing used by their clients, nor do they need to be aware of the data/meta-data distinction. All they need to do is to provide primitives of a keyed block storage:

- `put (key, data)` inserts the data block `data` and associates it with `key`, a block ID chosen by the data owner according to some naming scheme;
- `get (key)` returns the data associated with `key`.

This simple interface suffices to implement, on the data owner side, byte stream indexing and retrieval. Also, it is suitable to an environment in which service providers and users are mutually suspicious because it places as little burden as possible on the contributor side. The same approach was adopted by Venti [27] and by many peer-to-peer systems [2,9].

3.3. Backup Atomicity

Distributed and mobile file systems such as Coda [21] that support concurrent write access to the data and do not have built-in support for revision control differ significantly from backup systems. Namely, they are concerned about update propagation and reconciliation in the presence of concurrent updates. Not surprisingly, this approach does not adapt well to the loosely connected scenarios we are targeting: data owners are not guaranteed to meet *every* contributor storing data on their behalf in a timely fashion, which makes update propagation almost impossible. Additionally, it does not offer the desired atomicity requirement discussed in Section 2.2.

[1]In the sequel we use the terms "block ID", "name", and "key" interchangeably.

The *write once* or *append only* semantics adopted by archival [13,27], backup [9,29] and versioning systems [24,30] solve these problems. Data is always appended to the storage system, and never modified in place. This is achieved by assigning each piece of data an identifier that uniquely identifies it. Therefore, insertion of content (i.e., data blocks) into the storage mechanism (be it a peer machine, a local file system or data repository) is atomic. Because data is only added, never modified, the consistency guarantee is also met: insertion of a block cannot result in an inconsistent state of the storage mechanism.

A potential concern with this approach is its cost in terms of storage resources. It has been argued, however, that the cost of storing subsequent revisions of whole sets of files can be very low provided inter-version compression techniques like those described earlier are used [12,27,30]. In our case, once a contributor has finally transferred data to their owner's repository, it may reclaim all their storage resources, which further limits the cost of this approach.

From an end-user viewpoint, being able to restore an old copy of a file is more valuable than being unable to restore the file at all. All these reasons make the write-only approach suitable to the storage layer of our cooperative backup service.

3.4. Error Detection

As stated in Section 2.2, the storage mechanisms we are designing need to include error detection codes. Error-detecting codes can be computed either at the level of whole input streams or at the level of data blocks. They must then be part of, respectively, the stream meta-data, or the block meta-data. Here we argue the case of cryptographic hash functions as a means of providing the required error detection and as a block-level indexing scheme.

Cryptographic hash functions. The error-detecting code we use must be able to detect *malicious* modifications. This makes error-detecting codes designed to tolerate random, accidental faults inappropriate for our purposes. We must instead use *collision-resistant hash functions*, which were explicitly designed to tolerate malicious modifications. The *preimage-resistance* property of such functions is particularly important: given a hash value Y, it must be "hard" to find a message X such that the hash value of X is equal to Y [7].

Content-based indexing. Collision-resistant hash functions have been assumed to meet the requirements of a data block naming scheme as defined in Section 3.2.2, and to be a tool allowing for efficient implementations of single-instance storage [9,20,26,27,32,34,38]. In practice, these implementations assume that whenever two data blocks yield the same cryptographic hash value, their contents *are* identical. Given this assumption, the implementation of a single-instance store is straightforward: a block only needs to be stored if its hash value was not found in the locally maintained block hash table.

In [15], Henson argues that this assumption is not "risk-free". In particular, he argues that accidental collisions, although extremely rare, do have a slight negative impact on software reliability and yield silent errors. Given an n-bit hash output produced by one of the functions listed above, the expected workload to generate a collision out of two *random* inputs is of the order of $2^{n/2}$ [7], which indeed makes the risk of an accidental collision extremely low. More precisely, if we are to store, say, 8 GiB of data in the form of 1 KiB blocks, we end up with 2^{43} blocks, whereas SHA-1, for instance, would require 2^{80} blocks to be generated on average before an accidental collision occurs. In the framework of our fault-tolerance service, we consider this to be a reasonable guarantee since it does not impede the tolerance of faults in any significant way. Also, Henson's fear of *malicious* collisions does not hold given the preimage-resistance property provided by the commonly-used hash functions[2].

Content-addressable storage (CAS) thus seems a viable option for our software layer as it fulfills both the error-detection and data block naming requirements. CAS assumes a global block ID space shared across all users of the CAS provider. In [32], the authors conjecture that CAS providers could eventually become sufficiently ubiquitous that applications such as file systems could use them *opportunistically* without being dependent on any particular provider. In our scenario, CAS providers are contributors implementing the interface sketched in Section 3.2.2 and they do not trust their clients (data owners). Therefore, contributors need to enforce the block naming scheme, that is, make sure that the ID of each block is indeed the hash value of its content.

3.5. Encryption

Data encryption may be performed either at the level of individual blocks, or at the level of input streams. Encrypting the input stream *before* it is split into smaller blocks breaks the single-instance storage property at the level of individual blocks. This is because the encrypted output of two similar input streams will not be correlated since it may not be desirable.

Leaving input streams unencrypted and encrypting individual blocks yielded by the chopping algorithm does not have this disadvantage. More precisely, it preserves single-instance storage at the level of blocks at least *locally*, i.e., on the client side. If asymmetric ciphering algorithms are used, it does break the single-instance storage property *across* peers, because each peer encrypts data

[2] We are aware of the recent attacks found on SHA-1 by Wang et al. [36]. However, they do not affect the preimage-resistance of this function.

with its own private key. Solutions to this problem exist, notably *convergent encryption* [9]. However, we do not consider this a major drawback for the majority of scenarios considered where little or no data are common to several participants.

3.6. Backup Redundancy

Replication strategies. Redundancy management in the context of our collaborative backup service for mobile devices introduces a number of new challenges. Peer-to-peer file sharing systems are not a good source of inspiration given that they rely on redundancy primarily as a means of reducing access time to popular content [28].

For the purposes of fault-tolerance, statically defined redundancy strategies have been used in Internet-based scenarios where the set of servers responsible for holding replicas is known *a priori*, and where servers are usually assumed to be reachable “most of the time” [10,37]. Internet-based peer-to-peer backup systems [9,11,20] have relaxed these assumptions. However, although they take into account the fact that contributors may become unreachable, strong connectivity assumptions are still made: the inability to reach a contributor is assumed to be the exception, rather than the rule. As a consequence, unavailability of a contributor is quickly interpreted as the symptom of a malicious behavior [9,11].

The connectivity assumption does not hold in our case. Additionally, unlike with Internet-based systems, the very encounter of a contributor is unpredictable. This has a strong impact on the possible replication strategies, as well as on the techniques used to implement redundancy. In particular, *erasure codes* have been used as a means to tolerate failures of storage sites while being more storage-efficient than simple replication [37]. Usually, (n,k) erasure codes are defined as follows [22,37]:

- an (n,k) code maps a k-symbol block to an n-symbol codeword;
- $k + \varepsilon$ symbols suffice to recover the exact original data; the code is *optimal* when $\varepsilon = 0$;
- optimal (n,k) schemes allow the loss of $(n - k)$ symbols to be tolerated, and have an effective storage use of k/n.

In our attempt to improve storage efficiency while still maximizing data availability, such an approach seems very attractive.

However, as argued in [3,22,35], an (n,k) scheme with $k > 1$ can hinder data availability because it requires k peers to be available for data to be retrieved, instead of just 1 when mirroring (i.e., an $(n,1)$ scheme) is used. Also, given the unpredictability of contributor encounters, using a scheme with $k > 1$ is risky since a data owner may fail to store k symbols on different contributors. On the other hand, from a confidentiality viewpoint, increasing dissemination and purposefully placing less than k symbols on any given untrusted contributor may be a good strategy [10]. Intermediate solutions can also be imagined, for instance, mirroring blocks that have never been replicated and choosing a value of $k > 1$ for blocks already mirrored at least once. This use of different *levels of dispersal* was also mentioned by the authors of InterMemory [13] as a way to accommodate contradictory requirements. Finally, a dynamically adaptive behavior of erasure coding may be considered as [3] suggests.

Replica scheduling and dissemination. As stated in Section 2.2, it is plausible that a file will be only partly backed up when a newer version of this file enters the backup creation pipeline. On one hand, one could argue that the replica scheduler should finish distributing the data blocks from the old version before distributing those of the new version. This policy would guarantee, at least, availability of the old version of the file. On the other hand, in certain scenarios, users might want to favor freshness over availability. That is, they could request that newer blocks are scheduled first for replication.

This clearly illustrates that a wide range of *replica scheduling and dissemination policies and corresponding algorithms* can be defended depending on the scenario considered. At the core of a given replica scheduling and dissemination algorithm is a *dispersal function* that decides on a level of dispersal for any given data block. The algorithm must evolve *dynamically* to account for several changing factors. In FlashBack [23], the authors identify a number of important factors that they use to define a *device utility function.* Those factors include *locality* (i.e., the likelihood of encountering a given device again later) as well as *power and storage resources* of the device.

In addition to those factors, our backup software needs to account for the current level of trust in the contributor at hand. If a data owner fully trusts a contributor, for instance because it has proven to be well-behaved over a given period of time, the data owner may choose to store complete replicas (i.e., mirrors) on this contributor.

4. PRELIMINARY EVALUATION

This section presents our prototype implementation of the storage layer of the envisaged backup system, as well as a preliminary evaluation of key aspects.

4.1. Implementation Overview

We have implemented a prototype of the storage layer discussed above. As this layer is performance-critical, we implemented it in C. The resulting library, `libchop`, consists of 7 k of physical source lines of code. The library was designed in order to be flexible enough so that different techniques could be combined and evaluated. As

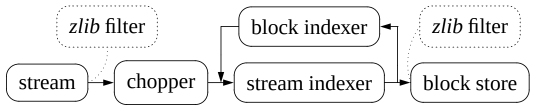

Figure 2. Data flow in the `libchop` backup creation pipeline.

such, the library itself consists of a few well-defined interfaces as shown in Figure 2. It comes with several implementations of each interface that were used in the experiments described below. The library itself is not concerned with the backup of file system-related meta-data such as file paths, permissions, etc. Implementing this is left to higher-level layers.

Implementations of the `chopper` interface chop input streams into small blocks, according to Manber's algorithm [25] or in fixed-sized blocks. Block indexers name blocks and store them in a keyed block stores (e.g., an on-disk database). Finally, the `stream_indexer` interface provides a method that iterates over the blocks yielded by the given chopper, indexes them, produces corresponding meta-data blocks, and stores them into a block store. In the proposed cooperative backup service, chopping and indexing are to be performed on the data owner side, while the block store itself will be realized by contributors.

`libchop` also provides *filters*, such as *zlib* compression and decompression filters, which may be conveniently reused in different places, for instance between a file-based input stream and a chopper, or between a stream indexer and a block store.

In the following experiments, the only stream indexer used is a "tree indexer" as shown in Figure 1. We used an on-disk block store that uses TDB as the underlying database [33]. For each file set, we started with a new, empty database.

4.2. Evaluation of Compression Techniques

Our implementation has allowed us to evaluate more precisely some of the tradeoffs outlined in Section 3. After describing the methodology and workloads that were used, we will comment the results obtained.

4.2.1. Methodology and Workloads

Methodology. The purpose of our evaluation is to compare the various compression techniques described earlier in order to better understand the tradeoffs that must be made. We measured both the storage efficiency and performance characteristics of each method. The measures were performed on a 500 MHz G4 Macintosh running GNU/Linux.

Name	Size	Files	Avg. Size
Lout (versions 3.20 to 3.29)	76 MiB	5853	13 KiB
Ogg Vorbis files	69 MiB	17	4 MiB
mbox-formatted mailbox	7 MiB	1	7 MiB

Figure 3. File sets.

We chose several workloads and compared the results obtained using different configurations. These file sets, shown in Figure 3, qualify as *semi-synthetic* workloads because they were not computer-generated but were not taken from an actual mobile device. The motivation for this choice was to purposefully target specific file *types*. The idea is that the results should remain valid for any file of these classes.

File sets. In Figure 3, the first file set contains 10 successive versions of the source code of the Lout document formatting system[1], i.e., low-density, textual input (C and Lout code). This workload is close to a set of active files as found on a Unix machine where the majority of files is smaller than 8 KiB and most file increases and modifications are of the order of 1 KiB [12]. Of course, it would be interesting to have file activity statistics on mobile devices such as a PDA or cell phone. Nevertheless, the results obtained on this workload are expected to be similar to those found with other textual data, such as XML. XML is currently used in an large number of desktop applications, and chances are that mobile applications may resort to XML as well.

The second file set shown in Figure 3 consists of 17 Ogg Vorbis[2] audio files, a high-density binary format. This is typical of the kind of data that may be found on devices equipped with sampling peripherals (e.g., microphone or camera whose output are lossy-compressed, be it in Ogg Vorbis, MP3, JPEG, etc.). The third file set consists of a single file: a mailbox in the Unix mbox format, that is, an append-only textual format. Such data are likely to be found in a similar form on communicating devices.

Configurations. Figure 4 shows the storage configurations we have used in our experiments. For each configuration, it indicates whether single-instance storage was provided, which chopping algorithm was used and what the expected block size was, as well as whether the input stream or output blocks were compressed using a lossless stream compression algorithm (*zlib* in our case). Configurations C_0 and C_0' serve as baselines for the comparisons. For the other configurations, we chose small block sizes that fit our requirements: 1 KiB of payload (on aver-

[1]See *http://lout.sf.net/*.

[2] Ogg Vorbis is a lossy audio compression format. See *http://xiph.org/*.

Config.	Single Instance?	Chopping Algo.	Expected Block Size	Input Zipped?	Blocks Zipped?
C_0	no	—	—	yes	—
C_0'	yes	—	—	yes	—
C_1	yes	Manber's	1024 B	no	no
C_2	yes	Manber's	1024 B	no	yes
C_3	yes	fixed-size	1024 B	no	yes
C_4	yes	fixed-size	1024 B	yes	no

Figure 4. Description of the configurations experimented.

age), along with TCP/IP headers and RPC additional data should yield packets slightly smaller than IEEE 802.11's MTU. Our implementation of Manber's algorithm uses a sliding window of 48 B which was reported to provide good results [26].

All configurations but C_0 used single-instance storage, realized using the `libchop` "hash" block indexer that produces an index handle based on the cryptographic hash of the block being indexed. For the experiments, 20-octet SHA-1 hashes were used. For C_0, a block indexer based on `libuuid` that systematically provides unique IDs (per RFC 4122 specifications) was chosen.

Our choice of configurations and file sets is quite similar to those described in [19,38], except that, as as explained in Section 3.1, we do not evaluate the storage efficiency of the delta encoding technique proposed therein. In addition, we propose an evaluation of computational costs, a critical criterion for our application.

4.2.2. Results

Figure 5 shows the compression ratios obtained for each configuration and each file set. The bars show the ratio of the size of the resulting blocks, *including* meta-data (sequences of SHA-1 hashes), to the size of the input data, for each configuration and each data set. The lines represent the corresponding throughputs.

Impact of the data type. Not suprisingly, the set of Ogg Vorbis files defeats all the compression techniques. Most configurations even incur a slight storage overhead due to the amount of meta-data generated. However, the two other data types are sensitive to the compression techniques being used.

Impact of single-instance storage. Comparing the results obtained for C_0 and C_0' shows benefits only in the case of the successive source code distributions of Lout, where it halves the amount of data stored (13 % vs. 26 %). This is due to the fact that successive versions of the software have a lot of files in common.

Inter-version compression. Considering the Lout file set, efficiency with C_1 is slightly better than that of C_0 (25 % vs. 26 %), but less efficient than the combination of

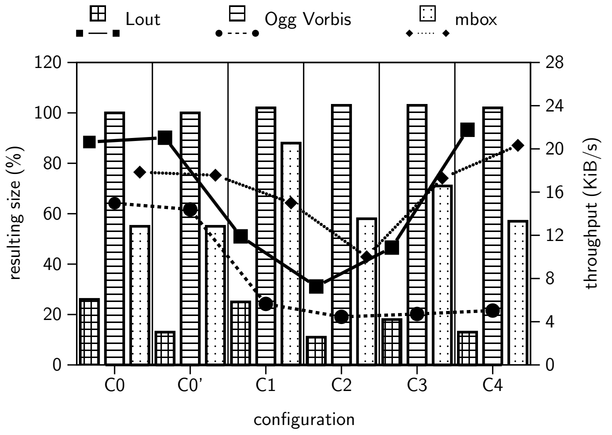

Figure 5. Storage efficiency and computational cost of several configurations.

zlib-based compression and per-file single-instance storage used in C_0'. As expected, C_2 (where the output blocks are compressed individually using *zlib*) provides the best compression ratio in the versioning case (the Lout file set). However, it is comparable to C_4 (11 % vs. 13 %) and even slightly worse for the two other file sets, although C_4 precludes single-instance storage at a sub-file level. One reason to this is that lossless compression algorithms yield better results when applied to larger input buffers, which compensates for the loss of sub-file single-instance storage.

The results in [38] are slightly more optimistic regarding the storage efficiency of a configuration similar to C_2. An explanation to this may be that the authors use a smaller block size, namely 512 B, and a larger file set. Additionally, it is unclear whether these measures include the size of meta-data.

Computational cost. A comparison between C_0 and C_0' shows that implementing single-instance storage using cryptographic hashes (SHA-1) incurs no significant overhead. The cost of *zlib*-based compression appears to be reasonable, particularly when performed on the input stream rather than on individual blocks, as evidenced, e.g., by C_3 and C_4.

Comparing C_2 and C_3 (where individual blocks are *zlib*-compressed) reveals that Manber's algorithm yields non-negligible computational overhead (throughput lowered by 33 %). While our implementation of Manber's algorithm could certainly be optimized, we believe that most of this overhead is intrinsic to Manber's algorithm which involves many fingerprint computations.

It is worth noting that the input data type has a noticeable impact on the computational cost. In particular, applying lossless compression techniques to the already-compressed Vorbis files is very costly computationally, while applying it to low-entropy data types proves to be much less costly. Therefore, it would be worth to disable *zlib* compression for compressed data types.

The configuration that offers the best tradeoff between computational cost and storage efficiency for low-entropy data is C_4, almost regardless of the type of input data.

5. CONCLUSION AND FUTURE WORK

In this paper, we have considered the viability of collaboration between peer mobile devices to implement a cooperative back-up service for mostly-disconnected operation. We have identified six essential requirements for the storage layer of such a service, namely: (i) storage efficiency; (ii) small data blocks; (iii) backup atomicity; (iv) error detection; (v) encryption; (vi) backup redundancy. The various design options for meeting these requirements have been reviewed and a preliminary evaluation carried out using a prototype implementation of the storage layer.

From a performance viewpoint, the execution overhead of implementing single-instance storage is negligible compared to that induced by the need to chop data into small blocks. Consequently, there is no compelling reason not to implement single-instance storage. This technique is likely to be beneficial in detecting intra-peer data commonality (e.g., versioning, duplicates); in more specific scenarios such as cooperative work, it could help detect inter-peer data commonality as well. We also conclude that, for the file sets considered, single-instance storage at a sub-file level yields little storage efficiency improvement compared to traditional lossless compression.

With respect to chopping into small blocks, Manber's algorithm, especially combined with block compression, provides a noticeable benefit in terms of storage efficiency in the case of subsequent versions of a set of files. However, it does not provide any significant improvement of storage efficiency when applied to a set of unrelated write-once files, and it incurs non-negligible computational overhead compared to simple fixed-size chopping. Therefore, we found that fixed-size chopping, combined with traditional lossless compression, may be preferable in the general case. Compressed media streams need to be treated specially, that is, lossless compression techniques should not be applied to them.

Future work on the optimization of the storage layer for the cooperative backup service concerns several aspects. First, the energy costs of the various design options need to be assessed, especially those related to the wireless transmission of backup data between nodes. Second, the performance and dependability impacts of various replica scheduling and dissemination strategies need to be evaluated as a function, for example, of the expected frequencies of data updates, cooperative backup opportunities and infrastructure connections. Third, it seems likely that no single configuration of the backup service will be appropriate for all situations, so dynamic adaptation of the service to suit different contexts needs to be investigated.

Finally, the issues relating to trust management, resource accounting and cooperation incentives need to be addressed, especially insomuch as the envisaged mode of mostly-disconnected operation imposes additional constraints on the more static wired peer-to-peer context. Current research in this direction, in collaboration with our partners in the MoSAIC project[1], is directed at evaluating mechanisms such as microeconomic and reputation-based incentives.

REFERENCES

[1] C. Batten, K. Barr, A. Saraf, S. Treptin. pStore: A secure peer-to-peer backup system. MIT-LCS-TM-632, MIT Laboratory for Computer Science, December 2001.

[2] K. Bennett, C. Grothoff, T. Horozov, I. Patrascu. Efficient Sharing of Encrypted Data. *Proc. of the 7th Australasian Conference on Information Security and Privacy (ACISP 2002),* (2384)pages 107–120, 2002.

[3] R. Bhagwan, K. Tati, Y-C. Cheng, S. Savage, G. M. Voelker. Total Recall: System Support for Automated Availability Management. *Proc. of the ACM/USENIX Symp. on Networked Systems Design and Implementation,* 2004.

[4] IEEE-SA S. Board. Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. 1999, *http://standards.ieee.org/*.

[5] W. J. Bolosky, J. R. Douceur, D. Ely, M. Theimer. Feasibility of a Serverless Distributed File System Deployed on an Existing Set of Desktop PCs. *Proc. of the Int. Conference on Measurement and Modeling of Computer Systems,* pages 34–43, 2000.

[6] G. Cobena, S. Abiteboul, A. Marian. Detecting Changes in XML Documents. *Proc. of the 18th ICDE,* pages 41–52, 2002.

[7] NESSIE Consortium. NESSIE Security Report. NES/DOC/ENS/WP5/D20/2, February 2003.

[8] L. Courtès, M-O. Killijian, D. Powell, M. Roy. Sauvegarde coopérative entre pairs pour dispositifs mobiles. *Actes des deuxièmes journées francophones Mobilité et Ubiquité (UbiMob),* pages 97–104, 2005.

[1]MoSAIC (Mobile System Availability Integrity and Confidentiality) is a prospective research project financed by the French national program for Security and Informatics (ACI S&I). Our partners are IRISA (Rennes) and Eurécom (Sophia Antipolis).

[9] L. P. Cox, B. D. Noble. Pastiche: Making backup cheap and easy. *Fifth USENIX OSDI*, pages 285–298, 2002.

[10] Y. Deswarte, L. Blain, J-C. Fabre. Intrusion Tolerance in Distributed Computing Systems. *Proc. of the IEEE Symp. on Research in Security and Privacy*, pages 110–121, 1991.

[11] S. Elnikety, M. Lillibridge, M. Burrows. Peer-to-peer Cooperative Backup System. *The USENIX FAST*, 2002.

[12] T. J. Gibson, E. L. Miller. Long-Term File Activity Patterns in a UNIX Workstation Environment. *Proc. of the Fifteenth IEEE Symp. on Mass Storage Systems*, pages 355–372, 1998.

[13] A. V. Goldberg, P. N. Yianilos. Towards an Archival Intermemory. *Proc. IEEE Int. Forum on Research and Technology Advances in Digital Libraries (ADL'98)*, pages 147–156, 1998.

[14] J. G. Hansen. The EDelta Binary Diff Algorithm. , *http://www.diku.dk/~jacobg/edelta/*.

[15] V. Henson. An Analysis of Compare-by-hash. *Proc. of HotOS IX: The 9th HotOS*, pages 13–18, 2003.

[16] J. J. Hunt, K-P. Vo, W. F. Tichy. An Empirical Study of Delta Algorithms. *Software configuration management: ICSE 96 SCM-6 Workshop*, pages 49–66, 1996.

[17] F. Junqueira, R. Bhagwan, K. Marzullo, S. Savage, G. M. Voelker. The Phoenix Recovery System: Rebuilding from the Ashes of an Internet Catastrophe. *Ninth HotOS*, 2003.

[18] M-O. Killijian, D. Powell, M. Banâtre, P. Couderc, Y. Roudier. Collaborative Backup for Dependable Mobile Applications. *Proc. of 2nd Int. Workshop on Middleware for Pervasive and Ad-Hoc Computing (Middleware 2004)*, pages 146–149, 2004.

[19] P. Kulkarni, F. Douglis, J. LaVoie, J. M. Tracey. Redundancy Elimination Within Large Collections of Files. *Proc. of the USENIX Annual Technical Conference*, 2004.

[20] M. Landers, H. Zhang, K-L. Tan. PeerStore: Better Performance by Relaxing in Peer-to-Peer Backup. *Proc. of the Fourth P2P*, pages 72–79, 2004.

[21] Y-W. Lee, K-S. Leung, M. Satyanarayanan. Operation-based Update Propagation in a Mobile File System. *Proc. of the USENIX Annual Technical Conference*, pages 43–56, 1999.

[22] W. K. Lin, D. M. Chiu, Y. B. Lee. Erasure Code Replication Revisited. *Proc. of the Fourth P2P*, pages 90–97, 2004.

[23] B. T. Loo, A. LaMarca, G. Borriello. Peer-To-Peer Backup for Personal Area Networks. IRS-TR-02-015, UC Berkeley; Intel Seattle Research (USA), May 2003.

[24] T. Lord. The GNU Arch Distributed Revision Control System. 2005, *http://www.gnu.org/software/gnu-arch/*.

[25] U. Manber. Finding Similar Files in a Large File System. *Proc. of the USENIX Winter 1994 Conference*, pages 1–10, 1994.

[26] A. Muthitacharoen, B. Chen, D. Mazières. A low-bandwidth network file system. *Proc. of the 18th ACM SOSP*, pages 174–187, 2001.

[27] S. Quinlan, S. Dorward. Venti: A new approach to archival storage. *Proc. of the First USENIX FAST*, pages 89–101, 2002.

[28] K. Ranganathan, A. Iamnitchi, I. Foster. Improving Data Availability Through Dynamic Model-Driven Replication in Large Peer-to-Peer Communities. *Proc. of the Workshop on Global and Peer-to-Peer Computing on Large Scale Distributed Systems*, pages 376–381, 2002.

[29] M. Rubel. Rsnapshot: A Remote Filesystem Snapshot Utility Based on Rsync. 2005, *http://rsnapshot.org/*.

[30] D. S. Santry, M. J. Feeley, N. C. Hutchinson, A. C. Veitch, R. W. Carton, J. Ofir. Deciding when to forget in the Elephant file system. *Proc. of the 17th ACM SOSP*, pages 110–123, 1999.

[31] M. Stemm, P. Gauthier, D. Harada, R. H. Katz. Reducing Power Consumption of Network Interfaces in Hand-Held Devices. *IEEE Transactions on Communications*, E80-B(8), August 1997, pages 1125–1131.

[32] N. Tolia, M. Kozuch, M. Satyanarayanan, B. Karp, T. Bressoud, A. Perrig. Opportunistic Use of Content Addressable Storage for Distributed File Systems. *Proc. of the USENIX Annual Technical Conference*, pages 127–140, 2003.

[33] A. Tridgell, P. Russel, J. Allison. The Trivial Database. 1999, *http://samba.org/*.

[34] A. Tridgell, P. Mackerras. The Rsync Algorithm. TR-CS-96-05, Department of Computer Science, Australian National University Canberra, Australia.

[35] A. Vernois, G. Utard. Data Durability in Peer to Peer Storage Systems. *Proc. of the 4th Workshop on Global and Peer to Peer Computing*, pages 90–97, 2004.

[36] X. Wang, Y. Yin, H. Yu. Finding Collisions in the Full SHA-1. *Proc. of the CRYPTO Conference*, pages 17–36, 2005.

[37] L. Xu. Hydra: A Platform for Survivable and Secure Data Storage Systems. *Proc. of the ACM Workshop on Storage Security and Survivability*, pages 108–114, 2005.

[38] L. L. You, K. T. Pollack, and D. D. E. Long. Deep Store: An Archival Storage System Architecture. *Proc. of the 21st ICDE*, pages 804–815, 2005.

SAUVEGARDE COOPÉRATIVE POUR DISPOSITIFS MOBILES

Ludovic Courtès[*]

Directeur de thèse : David Powell
Co-directeur de thèse : Marc-Olivier Killijian

Laboratoire d'accueil :
Laboratoire d'Analyse et d'Architecture du CNRS
7, avenue du Colonel Roche
31077 Toulouse CEDEX 4

Établissement d'inscription :
Institut National Polytechnique de Toulouse
6 allée Émile Monso - ZAC du Palays
BP 34038
31029 Toulouse CEDEX 4

Résumé

Nous présentons les fonctionnalités d'un système de sauvegarde coopérative pour dispositifs mobiles. Ce système repose sur la collaboration entre dispositifs pour assurer la sauvegarde et le recouvrement des données de chaque dispositif. Nous identifions les défis qui devront être relevés, en particulier dans le domaine du stockage réparti. Nous proposons une analyse de différents algorithmes de dissémination de données. Enfin, nous concluons sur nos premiers résultats et les axes de recherche à explorer.

Mots-clés

tolérance aux fautes, sauvegarde, mobilité, réseaux ad hoc, pair-à-pair

1 INTRODUCTION

Nous abordons la problématique liée à la conception d'un outil fournissant des mécanismes de sûreté de fonctionnement à des dispositifs mobiles dotés de moyens de communication sans fil. Nous considérons des dispositifs mobiles (ordinateurs portables, assistants personnels, etc.) n'ayant accès à une infrastructure fixe de communication (un réseau local ou Internet) que par intermittence. Ces dispositifs mobiles doivent être capables de communiquer entre eux, lorsqu'ils sont à proximité physique, en utilisant des moyens de communication sans fil. Cependant, nous souhaitons que l'outil développé, MoSAIC[1] soit utile à une large palette de systèmes, allant aussi bien de dispositifs très mobiles n'ayant que rarement accès à Internet, à l'autre extrême représenté par des machines connectées en permanence à Internet et peu ou pas mobiles. En outre, nous faisons l'hypothèse que les participants au service n'ont aucune relation de confiance au préalable.

MoSAIC a pour objectif de permettre aux dispositifs sur lesquels il s'exécute de tolérer les fautes logicielles ou matérielles pouvant entraîner la perte de données. Ces fautes sont le plus souvent permanentes : perte ou vol du dispositif mobile, effacement accidentel de données par l'utilisateur.À l'heure actuelle, les utilisateurs de systèmes mobiles effectuent le plus souvent la sauvegarde de leurs données uniquement lorsqu'ils ont accès à leur machine de bureau. Entre temps, il serait théoriquement possible d'utiliser l'accès à une infrastructure (par exemple UMTS ou GPRS) pour ce faire mais cette solution est très rarement utilisée, pour des raisons pratiques ou de coût.

[*] *<ludovic.courtes@laas.fr>*

[1] *Mobile System Availability, Integrity and Confidentiality, http://www.laas.fr/mosaic/.*

En revanche, nous pensons que l'avènement de dispositifs mobiles équipés de moyens de communication sans fil de courte portée offre des possibilités d'interactions *entre pairs* dont pourrait profiter un système de sauvegarde *coopératif*. En effet, l'utilisation répandue de systèmes mobiles communicant va permettre des interactions fréquentes mais de courte durée. Avec MoSAIC nous souhaitons tirer profit de ces interactions : à chaque rencontre de deux systèmes mobiles, le système de sauvegarde va automatiquement initier une demande de sauvegarde pour une partie de ses données, de manière *transparente*.

Nous présentons dans la section suivante les objectifs que devra atteindre MoSAIC, notamment en termes de sûreté de fonctionnement, en fonction des problèmes nouveaux qu'il pose. La section 3 présente nos travaux actuels portant sur l'évaluation analytique de la disponibilité des données obtenue grâce au service de sauvegarde. Enfin, dans la section 4, nous résumons nos résultats et présentons nos perspectives de recherche.

2 OBJECTIFS ET PROBLÉMATIQUES

Dans cette section, nous présentons les problèmes que nous souhaitons résoudre avec notre service de sauvegarde coopérative, en termes de sûreté de fonctionnement des dispositifs mobiles. Nous décrivons alors les fonctions fournies par le service. Enfin, nous montrons les nouveaux défis de sûreté de fonctionnement que doit résoudre ce service *coopératif*.

2.1 TOLÉRANCE AUX FAUTES DES DISPOSITIFS MOBILES

Les dispositifs mobiles, de par leur utilisation, sont sujets à des fautes permanentes (perte, vol, casse) et à des fautes transitoires (effacement accidentel, corruption). L'occurrence de ces fautes peut entraîner une perte des données stockées sur le dispositif. Or de tels dispositifs sont de plus en plus utilisés pour la saisie ou la capture de données nouvelles, dans un contexte où il est difficile voire impossible de réaliser des sauvegardes avec une fréquence raisonnable. Ce sont donc ces fautes que nous souhaitons pouvoir tolérer par le développement d'un service de sauvegarde.

L'objectif premier du service est donc d'améliorer la *disponibilité* des données stockées sur ces dispositifs mobiles. Chaque dispositif mobile peut sauvegarder ses données sur les dispositifs qui l'entourent, grâce à des moyens de communication sans fil, et de manière *opportuniste* (rôle d'utilisateur, *propriétaire* de données). En contrepartie, chaque dispositif doit également dédier un certain nombre de ses ressources de stockage au service pour que d'autres puissent en profiter de la même manière (rôle de *contributeur*). Nous faisons l'hypothèse que, dès qu'un contributeur a accès à Internet (c'est-à-dire dans une situation ou cet accès lui est peu coûteux), il transmet les données qu'il a acquises à leurs propriétaires. Ces derniers pourront alors les restaurer le cas échéant [3].

Il s'agit donc d'une approche semblable à celle des réseaux de partage de données *pair-à-pair* largement utilisés aujourd'hui sur Internet. Nous allons voir que cette approche, en particulier dans l'environnement mobile, lance des défis en termes de sûreté de fonctionnement.

2.2 DÉFIS

Nous faisons l'hypothèse que les participants à un tel service de sauvegarde n'ont *a priori* aucune relation de confiance entre eux. Par conséquent, nous devons prendre en compte la possibilité que des participants soient *malveillants* et cherchent à causer du tort à des participants individuels voire au service dans son ensemble. Par conséquent, un tel service doit pouvoir garantir la confidentialité et l'intégrité des données des utilisateurs, et doit aussi se prémunir contre les attaques en déni de services pouvant porter atteinte à son fonctionnement (rétention de données, inondation, égoïsme, etc.).

D'autres défis touchent au stockage et à la restauration des données : la dissémination des données sur des contributeurs multiples ainsi que le fait que les rencontres de contributeurs soient *imprévisibles* et *éphémères*. Enfin, le fait que les contributeurs soient potentiellement difficiles d'accès (par connexion directe ou *via* Internet) et suspects implique, de la part des propriétaires, de stocker les données avec un niveau de redondance approprié. Toutefois, compte-tenu du coût énergétique des communications sur de tels dispositifs, il est nécessaire de limiter autant que possible la quantité de données à échanger.

Dans cet article, nous nous focalisons sur les aspects ayant trait au stockage en nous intéressant à la conception d'un algorithme de dissémination des données.

2.3 SOLUTIONS

Le fait que les rencontres entre participants soient imprévisibles et potentiellement éphémères rend nécessaire la *fragmentation* des données à sauvegarder. En outre, ces fragments de données seront nécessairement disséminés, comme nous l'avons vu, avant, finalement, d'être rendus accessibles à leur propriétaire *via* Internet [3]. La dissémination peut par ailleurs être vue comme bénéfique du point de vue de la confidentialité des données [4].

Le niveau de redondance souhaité peut s'obtenir en stockant chaque fragment sur un certain nombre de contributeurs différents. En stockant n fois un fragment donné, il est alors possible de tolérer la défaillance (ou malveillance) de $n-1$ contributeurs. Cependant, pour un nombre de défaillances tolérées donné, il est possible de réduire la quantité de stockage nécessaire grâce à l'utilisation de *codes d'effacement* [11]. Schématiquement, il s'agit d'algorithmes qui, à partir d'une donnée d'entrée de taille k produisent $n > k$ fragments parmi lesquels n'importe quels k fragments suffisent pour reconstituer la donnée d'origine[1]. On tolère alors $n - k$ défaillances pour un coût de stockage de $\frac{n}{k}$. La réplication de fragments entiers peut donc être vue comme un cas particulier où $k = 1$. En pratique, les valeurs de k et n seront dépendantes de l'algorithme choisi (voir annexe).

Il faut cependant noter que le bénéfice en termes de disponibilité apporté par les codes d'effacement, pour un nombre de défaillances tolérées données, est fortement dépendant de la disponibilité des nœuds constituant le support de stockage [7]. En deçà d'une certaine disponibilité, il est préférable de faire appel à de la simple duplication. Par conséquent, il pourra être nécessaire de procéder à une adaptation dynamique du codage des copies en fonction de la disponibilité des contributeurs [1].

Plusieurs choix sont donc possibles pour paramétrer la fragmentation des données et plusieurs algorithmes de dissémination sont envisageables. Dans la section suivante, nous présentons différentes politiques possibles et proposons une méthode pour évaluer l'impact de ces algorithmes sur la disponibilité des données.

3 ÉVALUATION D'ALGORITHMES DE DISSÉMINATION DES DONNÉES

La section précédente a montré différents paramètres pouvant être pris en compte pour la dissémination et la redondance des données à sauvegarder. Dans cette section, nous présentons l'évaluation que nous souhaitons faire des algorithmes de dissémination envisageables ainsi que la méthodologie que nous souhaitons employer.

3.1 OBJECTIFS

S'agissant d'un service de sauvegarde, le principal critère d'évaluation sera bien entendu l'impact de l'algorithme sur la disponibilité des données à sauvegarder. Un *modèle* général du processus de sauvegarde doit donc être défini. Afin de simplifier ce modèle, on supposera que

[1]Cette description s'applique aux codes d'effacement *optimaux*.

dès lors que l'un des intervenants (propriétaire ou contributeur) a accès à Internet, alors les données sont « mises en sécurité ». En outre, nous considérons que dès lors qu'un propriétaire rencontre un contributeur il peut lui demander de stocker ses données. Ce modèle est composé de trois processus stochastiques Poissoniens à taux constants :

- le processus de rencontre d'un contributeur par le propriétaire de données, permettant à ce dernier de stocker une partie de ses données; ce processus a pour taux α;
- le processus de connexion à Internet de chaque participant, de taux β;
- le processus de défaillance des participants, de taux λ ; dans un premier temps, on fait donc abstraction des malveillances en les assimilant à des défaillances aléatoires.

À ce modèle nous ajoutons l'hypothèse suivante : un propriétaire de données n'est pas notifié de la défaillance de contributeurs disposant de ses données. C'est le cas le plus vraisemblable compte-tenu de la connectivité observée en environnement mobile. Par conséquent, nous considérons qu'un propriétaire de données ne pourra pas adapter sa politique de réplication et dissémination en fonction de la défaillance de ses contributeurs.

Ce modèle pourra bien sûr être raffiné par la suite. Pour chaque algorithme de dissémination donné, il pourra nous servir à évaluer analytiquement, en fonction de ces paramètres, les informations suivantes :

- la probabilité asymptotique que les données soient mises en sécurité (c'est-à-dire qu'elles atteignent Internet);
- le temps nécessaire aux données pour être rendues sûres (c'est-à-dire pour atteindre Internet);
- la disponibilité des données créées sur un dispositif mobile participant en fonction du temps.

Nous envisageons également d'utiliser ce modèle pour comparer différents algorithmes de dissémination en fonction de ces critères.

3.2 MÉTHODOLOGIE

Le processus de sauvegarde des données, dans le cadre du modèle simplificateur que nous venons d'évoquer, peut être modélisé sous la forme d'un graphe de Markov, c'est-à-dire un automate à états finis dont les branches sont étiquetées par les paramètres α, β et λ.

Supposons un algorithme de dissémination simple donnant une copie complète des données à chaque contributeur rencontré et se donnant pour objectif d'en distribuer N. La figure 1 représente les différents scénarios possibles dans l'exécution de cet algorithme, pour $N = 2$. On remarque deux états puits correspondant à la perte définitive des données (tous les dispositifs en disposant ont défailli) et à la « mise en lieu sûr » des données (un des dispositifs a accédé à Internet). Dans l'état initial noté 1/2, un seul exemplaire des données est disponibles (celui du propriétaire) et 2 copies restent donc à faire.

Dans l'état 3/0, trois exemplaires sont disponibles (dont celui du propriétaire) et il ne reste plus aucune copie à faire. À partir de l'état 2/1, il suffit qu'un dispositif parmi les 2 disposant des données accède à Internet pour passer dans l'état « sûr » (d'où le taux $2 \times \beta$); si le propriétaire défaille, alors on passe dans l'état 1/0 signifiant qu'un exemplaire est toujours disponible mais qu'aucune nouvelle copie ne sera faite à l'avenir; enfin, si le contributeur défaille, alors le propriétaire n'augmentera pas pour autant son nombre de copies souhaité, d'où le passage dans l'état 1/1.

Il s'agit là d'un modèle stochastique que nous pouvons généraliser à d'autres nombres de copies, à d'autres politiques, mais aussi à l'utilisation de codes d'effacement. La reformulation de ce modèle sous forme d'un réseau de Petri stochastique doit permettre de généraliser la modélisation à différentes politiques et paramètres pour le protocole de sauvegarde [2].

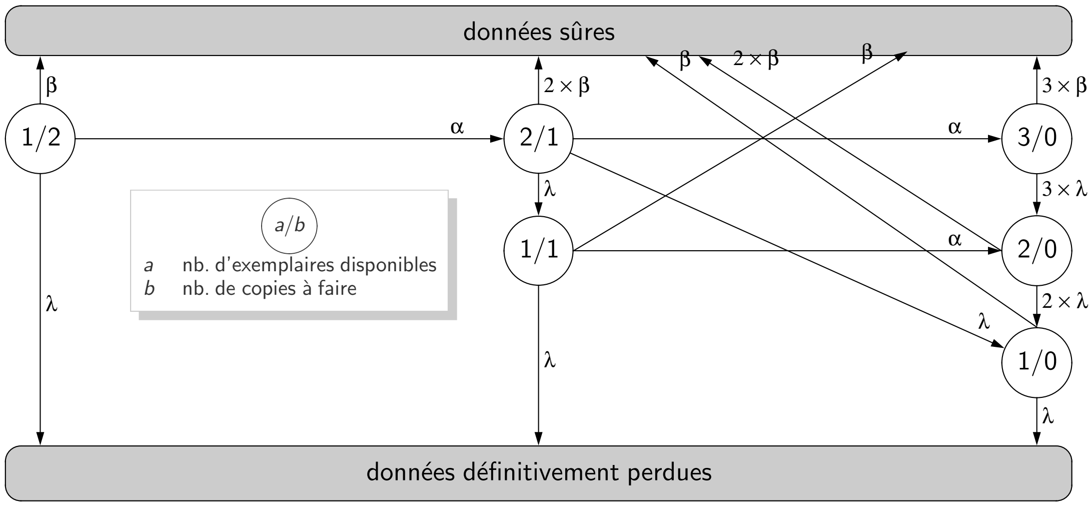

Fig. 1. Algorithme de dissémination réalisant 2 copies complètes des données originales.

4 CONCLUSION

Nous avons présenté le cadre dans lequel s'inscrit notre thèse, la sauvegarde coopérative pour dispositifs mobiles, ainsi que les défis nouveaux à relever dans ce contexte. Nous avons présenté, en particulier, le contour architectural du service que nous envisageons.

Nos travaux actuels portent sur l'évaluation analytique de la disponibilité des données offerte par un tel service de sauvegarde. Notre objectif est d'évaluer, pour des taux donnés de rencontre entre participants, d'accès à Internet, et de défaillance, la disponibilité atteinte en fonction du temps. Cette évaluation devrait nous permettre de considérer différentes politiques de dissémination des données.

RÉFÉRENCES

[1] R. Bhagwan, K. Tati, Y-C. Cheng, S. Savage, G. M. Voelker. Total Recall : System Support for Automated Availability Management. *Proc. of the ACM/USENIX Symp. on Networked Systems Design and Implementation*, 2004.

[2] C. Béounes, M. Aguéra, J. Arlat, K. Kanoun, J-C. Laprie, S. Metge, S. Bachmann, C. Bourdeau., J-E. Doucet, D. Powell, P. Spiesser. SURF-2 : A Program for Dependability Evaluation of Complex Hardware and Software Systems. *Proc. of the Twenty-Third IEEE Annual Int. Symp. on Fault-Tolerant Computing*, pages 668–673, 1993.

[3] L. Courtès, M-O. Killijian, D. Powell, M. Roy. Sauvegarde coopérative entre pairs pour dispositifs mobiles. *Actes des deuxièmes journées francophones Mobilité et Ubiquité (UbiMob)*, pages 97–104, 2005.

[4] Y. Deswarte, L. Blain, J-C. Fabre. Intrusion Tolerance in Distributed Computing Systems. *Proc. of the IEEE Symp. on Research in Security and Privacy*, pages 110–121, 1991.

[5] C. Huang, L. Xu. STAR : An Efficient Coding Scheme for Correcting Triple Storage Node Failures. *Proc. of the Fourth USENIX FAST*, pages 197–210, 2005.

[6] R. Katti, X. Ruan. S-Code : New Distance-3 MDS Array Codes. *IEEE Int. Symp. on Circuits and Systems*, 2005.

[7] W. K. Lin, D. M. Chiu, Y. B. Lee. Erasure Code Replication Revisited. *Proc. of the Fourth P2P*, pages 90–97, 2004.

[8] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, D. A. Spielman. Efficient Erasure Correcting Codes. *IEEE Transactions on Information Theory*, 47(2), February 2001, pages 569–584.

[9] P. Maymounkov. Online Codes. TR2002-833, Secure Computer Systems Group, New York University, NY, USA, November 2002.

[10] M. Mitzenmacher. Digital Fountains : A Survey and Look Forward. *Proc. of the IEEE Information Theory Workshop*, pages 271–276, 2004.

[11] L. Xu. Hydra : A Platform for Survivable and Secure Data Storage Systems. *Proc. of the ACM Workshop on Storage Security and Survivability*, pages 108–114, 2005.

[12] L. Xu, V. Bohossian, J. Bruck, D. G. Wagner. Low Density MDS Codes and Factors of Complete Graphs. *IEEE Transactions on Information Theory*, 45(1), November 1999, pages 1817–1826.

ANNEXE : ALGORITHMES DE CODES D'EFFACEMENT

On utilise la notation (n,k) pour désigner un code d'effacement qui, à partir d'une donnée partitionnée en k fragments, produit n fragments, avec $n > k$. Un code d'effacement est dit *optimal* si le nombre de fragments nécessaire pour recouvrer la donnée d'origine est égal à k ; on dit qu'il est quasi-optimal si il suffit de $k + \varepsilon$ fragments, où ε est petit par rapport à k [12]. On appelle *taux* d'un code le rapport entre le nombre de fragments nécessaires pour reconstituer la donnée initiale et le nombre n de fragments produits. On appelle *distance* d'un code la différence entre le nombre de fragments produits n et le nombre de fragments nécessaires, plus 1. Par exemple, dans le cas d'un code optimal, la distance est $d = n - k + 1$ [5].

En pratique, un certain nombre de codes d'effacement sont prévus pour un taux fixe. On notera par exemple les codes optimaux suivants :

Nom	**Distance**	**Taux**
B-Code [12]	3	$\frac{k}{k+2}$
STAR [5]	4	$\frac{k}{k+3}$
S-Code [6]	3	$\frac{k}{k+2}$
Tornado [8]		au choix, choisi *a priori*

Comme on le voit, les codes Tornado définissent tout une famille de codes dont le taux doit être choisi avant utilisation. Par ailleurs, il existe des codes dits « sans taux » capables de produire potentiellement une infinité de fragments codés parmi lesquels seuls $k + \varepsilon$ suffisent pour restaurer la donnée d'origine [9,10].

Increasing Data Resilience of Mobile Devices with a Collaborative Backup Service

Damien Martin-Guillerez
IRISA/ENS-Cachan
Campus de Beaulieu, 35 042 Rennes Cedex, FRANCE
dmartin@irisa.fr

Abstract

Whoever has had his cell phone stolen knows how frustrating it is to be unable to get his contact list back. To avoid data loss when losing or destroying a mobile device like a PDA or a cell phone, data is usually backed-up to a fixed station. However, in the time between the last backup and the failure, important data can have been produced and then lost.

To handle this issue, we propose a transparent collaborative backup system. Indeed, by saving data on other mobile devices between two connections to a global infrastructure, we can resist to such scenarios.

In this paper, after a general description of such a system, we present a way to replicate data on mobile devices to attain a prerequired resilience for the backup.

1. Introduction

The production of sensible data on mobile devices, such as PDAs or mobile phones, has increased with the use of such devices. The loss of those data can have painful consequences for the user (e.g. loss of phone numbers or meeting notes).

To reduce data loss, many devices have a synchronization mechanism. The main issue in synchronization is the necessary proximity of the user and his computer and thus, there is a time period during which device failure means irreversible data loss. For example, if you take a note on your PDA during a meeting and this PDA gets lost on your way home then the note is definitely lost. However, more and more mobile devices come with wireless connectivity like IEEE 802.11. Thus, using neighbor devices to save data right after its production can decrease data loss. Data can be restored either from a global-scale network like the Internet or directly from the backup device.

We aim at creating a transparent collaborative backup service for mobile devices [5]. Such a service needs to meet specific requirements outlined in section 2. Then, we analyze several specific issues of mobile device data and replication in section 3. Afterwards, we present a way to order replicas in that system in section 4. After presenting existing systems in section 5, we outline works that are still pending and conclude in section 6.

2. Design overview

Our main aim is to design a transparent backup system that can handle high mobility. Thus, it needs to handle two scenarios: (i) when connected to a global network like the Internet, the system must use the opportunity to save data on a resilient server and (ii) when disconnected from the global network, it must use neighbor terminals to backup selected data.

Depending on data production (e.g. production rate, data importance), the system should adapt the level of replication. Moreover, the system needs to be protected against egoistic participants that backup but do not provide resources to others. Of course, we want the system to be as transparent for the user as possible. That means that very few actions are required from the user during the backup or the restore and that neither prior trust relationship with other peers nor extra hardware is required.

We consider only following backup scenarios: a client terminal can either backup its data to another terminal or to an Internet server. Later, the backup peer can save the data on the Internet server. If a failure occurs, the client terminal can restore the data from the peer or from the server. We do not consider propagating backup through peers because:

- Propagating backups costs energy and others resources that will not be available for further backups.
- Only a full replica can be issued in such situations, contrary to considered backup dispersion schemes.
- Only the client terminal can know when it's necessary to issue a replication. A replication issued by a

backup terminal has a good chance to be useless.

3. Data issues

Mobile device data. We will now look at data produced on classical mobile devices and at their attributes to outline related issues. The size mainly depends on data type (from less than 200 bytes for SMS to hundred of megabytes for movies). The second attribute is data importance, high for notes taken during a meeting to very low for holiday pictures for instance. Dependencies are also important: a SMS may be useless without all preceding SMS in a discussion thread. When a data item depends on preceding data, we call it a *temporal dependency*. On the contrary, when a data item is interdepent with others data, we call it a *spacial dependency*. So, mobile device data are categorized by size, dependencies and importance.

The size affects the backup system in means of (i) resisting to mobility or network problems during a transmission and (ii) avoiding monopolizing one terminal memory. Dependencies affect backup integrity and thus the system needs some version tracking. Finally, we assign a priority for each data item relatively to its importance and try to save data items with highest priority first.

Dispersion of replicas. File fragmentation is imposed by potentially high sized data. Moreover, the risk of a terminal not correctly restoring a replica creates a need for a flexible replication scheme. Courtes et al. [4] have already defined the redundancy and compression methods we use. We consider that all data items with spatial dependencies are agglomerated into one (the priority of the new item is the highest of the agglomerated items). Then, we consider the (n, k) replication scheme that fragments the data item into n pieces where only k are required for reconstruction. We also consider delta-compression which saves only the changes between two versions of the same file. Replication when delta-compressing is made on generated delta.

So, we consider that every data item to save follow this format: n fragments and only k needed to reconstruct the data item, possible temporal dependencies (the priority of old data should be increased if it is lower than the priority of the new data).

Version tracking. Given those propagation and dissemination schemes, some issues can appear regarding arrival of new version of a data item to backup. First, the old version of a data should be kept until all dependencies of the new version have been backed-up to the resilient server. Moreover, conflicts may appear in our system. When you backup the data of a mobile device on a fixed station, no conflict appears since all new versions of a data item are created on this device. But, with our propagation scheme, a conflict may appear (cf. figure 1) when a data item is backed-up on another mobile device and an old version of this data item is located on the Internet server. If a failure occurs in this case, the client may restore the old version from the server and work on it, generating a conflict with the version backed-up on the mobile device. In such a situation, our system must use conflict resolution mechanisms like in Bayou [9].

4. Maximizing backup reliability

We will start studying the (n, k) replication scheme. Let $\mathbb{P}_i$ be the probability of getting back the replica i and $\mathbb{P}_i^l$ the probability of being able to get back l replicas between the first i ones. Then we have (cf. figure 2):

$$\mathbb{P}_i^l = (1 - \mathbb{P}_i) \cdot \mathbb{P}_{i-1}^l + \mathbb{P}_i \cdot \mathbb{P}_{i-1}^{l-1} \quad (1)$$

So, when backing-up an additional replica, we can estimate the impact on the probability of getting back the entire data item. That is right, of course, only if each replica is saved on a different terminal. We can handle the case of two replicas being backed-up on the same terminal by assuming that they have the same probability $\mathbb{P}_i$. Thus, if we save m replicas onto the same terminal at the same time, we have:

$$\mathbb{P}_{i+m}^l = (1 - \mathbb{P}_{i+1}) \cdot \mathbb{P}_i^l + \mathbb{P}_{i+1} \cdot \mathbb{P}_i^{l-m} \quad (2)$$

We assume that a further save of a replica on an already used terminal is an independent event. Finally we must take into account the temporal dependencies. The probability of restoring correctly a new data item that depends on old ones is the probability of restoring the new data item multiplied by the probability of restoring the old data.

We said in section 3 that each data item is associated with a priority given as a desired backup resilience (e.g. a probability). A prior mechanism should have established this priority. Hence we can order data items to be backed-up using a queue ordered by the priority minus the computed probability of successful backup.

Thus, we get a general algorithm to order data packets to save. First, we try to save while the peer is reachable. The first data item that may be saved on this peer is pulled off the queue. We try to save the next packet of this item and recompute the probability of a successful backup. If the probability is too low, the data item is inserted back into the queue. We use (1) and (2) to recompute the probability.

5. Related works

Usually, data resilience is increased using hardware replication. In network file systems, replication of data uses several data servers [8]. Peer-to-peer file systems have used replication to increase data availability [3] and have paved the way for collaborative backups [1].

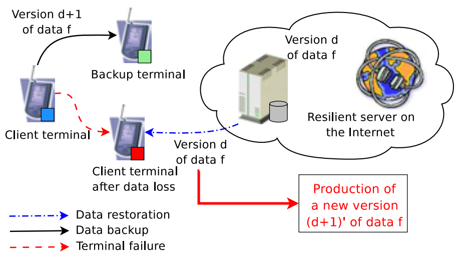

Figure 1. Conflict during a restoration.

In a mobile context, Roam [7] uses peers to replicate data for high availability but can hardly handle high mobility due to the clusterization of the peers. Data replication between clusters is needed when a peer switches clusters. In fact, Roam is not designed for backup recovery but for high availability and data sharing. Moreover, Roam does not exploit opportunistic replication on random mobile peer. AdHocFS [2], another file system focused on high availability and data sharing by transposing peer-to-peer file systems' paradigms to ad hoc networks, does not give support for high mobility. FlashBack [6], a backup system for Personal Area Network (PAN), can efficiently handle data loss. FlashBack is designed for people that wear several wireless mobile devices. Thus, FlashBack suffers from the same limitations as AdHocFS and Roam.

6. Conclusion

We have presented a general design for a backup system that can handle high mobility and that do not rely on pre-established relationships. Issues regarding incentives, confidentiality, high mobility and resource management are still to be resolved.

Indeed, several requirements have been outlined in section 2: the system must be transparent, should not rely on prior relationships nor on specific hardware. The system must automatically assign the priority to the data, should use incentives and confidentiality techniques. The network layer should use classical wireless interface without interference with their classical uses.

A main pending issue is the probability estimation of one packet to be correctly restored ($\mathbb{P}_i$). The main parameter is the reliability of the device itself eventually given by incentives. Other parameters can be battery lifetime, terminal context and memory availability. Resource management implies deletion of replicas to free memory on backup terminals. We need to know which replicas to delete and the impact on the system efficiency. Our future works will concentrate on those issues related to resource management.

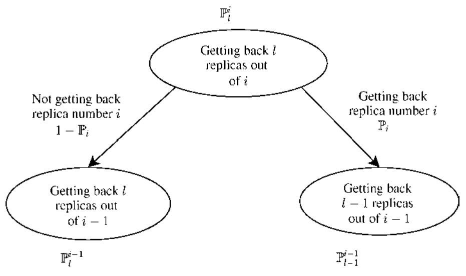
$\mathbb{P}_l^i$
Getting back l replicas out of i
Not getting back replica number i
$1 - \mathbb{P}_i$
Getting back replica number i
$\mathbb{P}_i$
Getting back l replicas out of $i-1$
Getting back $l-1$ replicas out of $i-1$
$\mathbb{P}_l^{i-1}$
$\mathbb{P}_{l-1}^{i-1}$

Figure 2. Graphical proof of equation (1).

References

[1] C. Batten, K. Barr, A. Saraf, and S. Treptin. pStore: A Secure Peer-to-peer Backup System. Technical Report MIT-LCS-TM-632, MIT Laboratory for Computer Science, Dec. 2001.

[2] M. Boulkenafed and V. Issarny. AdHocFS: Sharing Files in WLANS. In *The Second IEEE International Symposium on Network Computing and Applications (NCA'03)*, Apr. 2003.

[3] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *The Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, July 2000.

[4] L. Courtès, M.-O. Killijian, and D. Powell. Storage Tradeoffs in a Collaborative Backup Service for Mobile Devices. To appear, 2006.

[5] M.-O. Kilijian, D. Powell, M. Banâtre, P. Couderc, and Y. Roudier. Collaborative Backup for Dependable Mobile Applications. In *The 2nd International Workshop on Middleware for Pervasive and Ad-Hoc Computing (Middleware)*. ACM, Oct. 2004.

[6] B. T. Loo, A. LaMarca, and G. Borriello. Peer-To-Peer Backup for Personnal Area Networks. Technical Report IRS-TR-02-015, Intel Research Seattle - University of California at Berkeley, May 2003.

[7] D. Ratner, P. Reiher, and G. J. Pope1. Roam: A Scalable Replication System for Mobile Computing. In *The Workshop on Mobile Databases and Distributed Systems (MDDS)*, pages 96–104, Sept. 1999.

[8] M. Satyanarayanan. Scalable, Secure and Highly Available Distributed File Access. *IEEE Computer*, 23(5):9–21, May 1990.

[9] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *The 15th ACM Symposium on Operating Systems Principles (SOSP'95)*, Dec. 1995.