

Designing an NFS-based Mobile Distributed File System for Ephemeral Sharing in Proximity Networks

Nikolaos Michalakis
New York University
719 Broadway, Room 715
New York, NY 10003
nikos@cs.nyu.edu

Dimitris N. Kalofonos
Nokia Research Center
5 Wayside Road
Burlington, MA 01803
dimitris.kalofonos@nokia.com

Abstract— This paper addresses the need for ephemeral file sharing in proximity networks and explores the necessary requirements that a mobile distributed file system (M-DFS) should satisfy in order to operate properly under typical conditions of such networks. We chose an M-DFS design based on the widely deployed NFS because of its ease of portability, lightweight model and its use in the past as a basis for building distributed file systems. To identify NFS shortcomings in proximity networks using Bluetooth and 802.11, we measured its performance under various bandwidth and delay combinations, network disconnections, low signal reception zones and in the presence of packet loss. Based on our observations, we present some proposals for the design of an M-DFS suitable for ephemeral sharing and describe a possible prototype implementation using an NFS loopback server approach.

I. INTRODUCTION

Today, users can use their mobile phones to store and share personal content, such as photos and videos. However, each mobile application has its own method of sharing its data and usually through file transfer, not file access. As mobile devices become more sophisticated and interact more with the environment in their proximity, there is an increasing need for sharing mobile data via a file system interface, i.e. via a mobile distributed file system (M-DFS). Our focus is on *ephemeral* file sharing, which involves spontaneous and short-lived collaboration among users.

In this paper we explore NFS v3 [1] as the basis of a M-DFS for ephemeral sharing. NFS is one of the most successful DFS in use today and it has been widely used and tested on many different platforms. An NFS-based file system can enjoy wide portability, which has made it attractive as a basis for other file systems with additional features [2], [3], [4]. NFS also appears to be a good choice as a basis for short-term sharing, because it has a lightweight mounting and unmounting process that would allow mobile devices to discover other file systems and mount them on demand. Using other popular DFSs such as Coda or Intermezzo [5], [6] as a basis of an M-DFS for ephemeral sharing is a more difficult task. These file systems have extra complexity intended for *persistent* sharing. Ephemeral file sharing has weaker requirements, so features such as disconnected operation are not necessary and do not justify the additional complexity and size of the system,

especially for light-weight handheld devices. In general, it is more difficult to remove features when modifying a system than to add, because of all the interconnections that are broken and need to be addressed.

We believe there is a need for an M-DFS for ephemeral sharing in proximity networks that is lightweight, portable and extensible in order to accommodate user needs, while imposing minimal resource overhead to the handheld device. Although the existing file systems provide solutions to the issue of disconnected operation, the focus has been more on persistent file sharing. In this paper we make the case for ephemeral sharing in the mobile environment and give examples of user cases and requirements. As a first design step, we examine the performance and reveal the shortcomings of standard NFS in wireless proximity networks. Although a study of NFS in wireless networks can be found in [7], [8], it focuses mainly on the effects of packet loss in infrared links, so these results were found inadequate for our purposes. The results we present in this paper involve a wide range of network conditions using measurements from actual Bluetooth and 802.11b links, as well as using a simulation model based on [9]. Motivated by these results, we outline some design challenges that need to be addressed to create an M-DFS for ephemeral file sharing. Finally, we propose an easy-to-prototype user-level implementation.

The rest of the paper is organized as follows: Section II presents related work; Section III distinguishes ephemeral and persistent file sharing; Section IV provides a performance evaluation of the current NFS standard (version 3) in wireless proximity networks; Section V outlines our design suggestions and a possible implementation approach; and Section VI presents our conclusions and future directions.

II. RELATED WORK

Coda [5] is the first file system to support true disconnected operation. The idea introduced by Coda was that client and server cooperate to allow a user to access files even when disconnected from the network. The client performs a form of aggressive caching named hoarding, where whole files are permanently cached. A lot of work in Coda has been done to allow disconnected operation when the network is not

available and to try to optimize network utilization when there is intermittent connectivity by reducing the size of the data passed between client and server. The Intermezzo file system [6] is a lightweight descendant of Coda that focuses on the aspects of disconnected operation and intermittent connectivity. Aspects of Coda’s design on intermittent connectivity apply to ephemeral file sharing as well and need to be considered in an M-DFS design.

NFS/M [10] is an attempt to modify NFS to have the same functionality as Coda. The kernel was modified such that the client can support disconnected operation via data prefetching, server emulation, and aggressive caching. Although it presents an example of how NFS can be modified to deal with intermittent connectivity, the design requires kernel modifications and is intended for persistent file sharing.

A different approach is to create file systems on top of standard NFS as opposed to modifying it. The Self-certifying file system (SFS) [3] was designed to span the Internet while maintaining high security guarantees both for clients and servers. SFS was built using NFS v3 as the basis for RPC communication and access to local file systems. This set of tools evolved to the SFS toolkit [2]. The Low Bandwidth file system (LBFS) [4] was designed using the SFS toolkit to provide remote access to files at a good performance even under low bandwidths. It uses a combination of RPC compression, aggressive caching and file difference propagation to reduce the traffic over the network. However, most of the performance gains of LBFS are limited to applications that handle text and when small byte differences are propagated to the server.

III. THE CASE FOR EPHEMERAL FILE SHARING IN THE MOBILE ENVIRONMENT

In this Section we present examples of user scenarios for ephemeral file sharing. They illustrate the application-level requirements for the design of an M-DFS. With these examples in mind we then describe the differences between ephemeral and persistent file sharing. Figure 1 illustrates the distinction between the two types of file sharing.

A. Examples of User Scenarios Illustrating Requirements

- Hermes and Sophia are taking a tour in a museum, while viewing historical data at their mobile phones made available from their guide’s PDA tour notes. After the tour is over they wonder away from the group and after a few minutes their phones automatically “forget” the sharing session.
- Hermes and Sophia are sharing pictures from their camera phones. Hermes needs to take a call in private, so he moves away. Sophia can keep browsing the images she has seen so far without problems and can look at new pictures when Hermes returns, without any reconfiguration.
- Hermes places his portable hard-drive file server at his locker and is listening to his MP3s from his phone while exercising at the next room. Without an M-DFS,

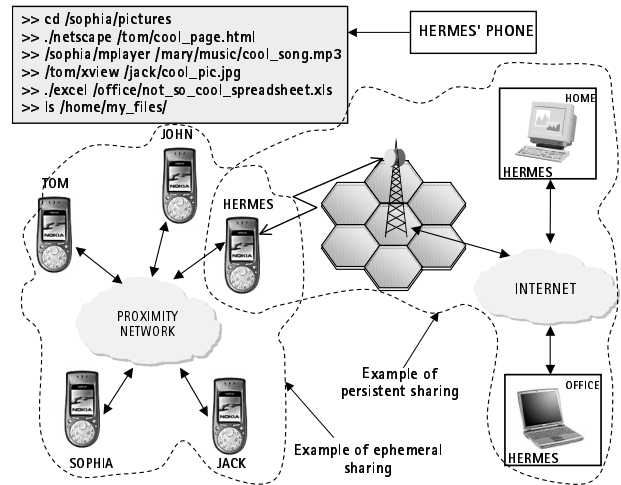


Fig. 1. Ephemeral versus persistent file sharing.

disconnections due to bad signal would cause him to restart his player often.

- Sophia mounts Hermes’ public games directory and plays with his Tetris during the flight to Helsinki. After they part she has no more access to the game.
- While at lunch Hermes decides to export to Sophia his pictures from his trip to Miami and to his supervisor the documents from his business meeting. Sophia should not see the documents from the business meeting and his supervisor should not see the pictures from Miami.
- Proximity applications enable a user to discover and mount a host file system to: read the newspaper provided by the subway mobile network in PDF; have a look at the trailers in a movie theater before choosing a movie; listen to samples on her mobile phone in the music store before buying a CD.

B. Ephemeral vs. Persistent File Sharing

In *ephemeral file sharing* a remote file system is discovered and mounted to enable short-lived collaboration among users. Clients use caching mainly to improve performance (e.g. in low bandwidth-links), not to guarantee disconnected operation. An M-DFS for ephemeral sharing deals with short disconnections (order of seconds-minutes) that occur due to intermittent connectivity at the link level and no state is expected to survive long disconnections; if files are to remain accessible in the long-term, the user needs to “save” local copies. Finally, ephemeral sharing is intended mainly for read-only file sharing.

On the other hand, *persistent file sharing* enables users to keep a personal distributed file repository among their devices (e.g. phones, PCs) using a persistent file server usually attached on a fixed network. Clients are likely to perform

“hoarding” when fully connected, they can operate on files when disconnected, and perform reintegration-conflict resolution when reconnected. Because of aggressive whole file caching files are accessible in the long-term without explicitly “saving” local copies. Disconnections in this case can be arbitrarily long (order of hours-days). Finally, persistent sharing is usually intended for read-write file sharing.

IV. NFS SHORTCOMINGS IN PROXIMITY NETWORKS

In this Section, we examine the performance of NFS v3 (found in the kernel of Linux) in Proximity Networks. This is the first step we have taken towards designing an M-DFS for ephemeral file sharing based on NFS. Our aim was to identify NFS shortcomings in the mobile environment. We performed measurements using an actual wireless experimental setup (Bluetooth v1.1 PANs and 802.11b) and some using simulated client-server wireless links.

A. Problems with NFS and Mobility

We conducted experiments causing IP address changes and disconnections to NFS Linux 2.4.18 clients and servers to observe any problems. In these experiments a client had mounted a remote file system or was about to mount one after the server had already started. Table I summarizes the problems encountered.

B. Performance Results

For our wireless experimental setup we used two laptops running NFS v3 in Linux 2.4.18 kernels. For 802.11b measurements the laptops were connected to the same access point. For IP over Bluetooth we used the BlueZ Linux stack with the default configuration and Bluetooth v1.1 USB adapters. We also simulated wireless links using Ethernet links between the client and the server via a third routing host which was running a user-level Click software router [9]. The router gets copies of incoming packets from one interface and sends them to the other after processing them to control delay, bandwidth, disconnections and losses.

We first examined how changing the bandwidth affects the performance of NFS. Figure 2 (left) illustrates the measurements using Bluetooth and 802.11b. There is a notable difference between Bluetooth and 802.11b when both have approximately the same bandwidth. This is due to the difference in round trip times (RTT), which were approximately 27 ms for Bluetooth and 4 ms for 802.11b. If we assume that the size of the request RPCs is negligible compared to the size of the replies when reading files and that RPCs are sent synchronously, then the time T to completion can be written as $T \approx N \times (RTT + (rpc/bw))$, where N is the total number of RPCs sent, bw is the available bandwidth and rpc is the average reply RPC size. We see from this simple equation that for high bw , the term RTT dominates the performance and for low bw the RPC transmission time term rpc/bw dominates. Figure 2 (right) illustrates measurements using simulated wireless links (this allowed more flexibility in setting the parameters) and supports our simple analysis. The

above results indicate that just increasing the bandwidth of a wireless access technology may not necessarily improve the performance of a synchronous request-reply application such as NFS.

In the default configuration a Bluetooth transmitter keeps retransmitting link-level packets until the receiver returns a positive acknowledgement. This introduces very high delays and delay variations as the distance between receiver and transmitter increases due to user mobility. Taking the average of 100 pings we measured $RTT_{mean}(std)$ ranging between 24(5) ms, when the two nodes were next to each other, and 609(277) ms close to the edge of the coverage. To examine the effect of large delays on NFS depending on the selected transport parameters, we measured a 50K file read time using a 75KB/s Bluetooth link at a zone with low signal reception (we called it “dark” zone), where the RTT was approximately 220(120) ms. The results are illustrated in Figure 3. We observe that in the case of high delays, when UDP is used the longer the NFS timeout (**timeo**) the better the performance. Because of the stop and wait retransmission mechanism in Bluetooth, unnecessary RPC retransmissions slow down the read performance, as more packets are added to the Bluetooth FIFO send buffer. With TCP transport there are no NFS-level retransmissions, but transport-level retransmissions are known not to interact well with link-level retransmissions and the observed performance is in the middle of the observed range.

Finally, we wanted to examine the effect of temporary disconnections and random packet losses (losses occurred only in 802.11b, not in Bluetooth) on NFS performance. First we noticed that, although NFS clients take advantage of the kernel’s page caching (as do native file systems) when reading files, if ACCESS and LOOKUP RPCs are sent during a disconnection the file cannot be read, even though it may be already entirely cached in the client (warm cache). NFS caching helps much less against disconnections in the case of directory listing, because even though the GETATTR and REaddir RPCs can be cached for a short period (typically 30 sec), the ACCESS and LOOKUP RPCs have to be transmitted. To see the effect of disconnections in this case, we measured the time to list a 300-entry directory using a fixed disconnection pattern with three 5.5s disconnections each separated by 20 seconds. The results with disconnections are presented in Figure 4 (left). We see that both with UDP (timeout was set to 1 sec) and TCP the performance suffers because of the time wasted right after the connectivity is restored due to the exponential back-off algorithms used. Also note that TCP resumes more slowly after each disconnection which is perceived as congestion. The effect of random packets dropped is again time unnecessarily wasted. This time, though, the effect is more dramatic when UDP is used, because an entire timeout period (which can be set to several seconds) is wasted for every single packet dropped, even though connectivity may be restored instantaneously after each random loss. Measurements using our simulation model in Figure 4 (right) illustrate this.

TABLE I
NFS PROBLEMS WITH CLIENT/SERVER MOBILITY.

	Client has mounted	Client not mounted (server already started)
Client IP address change:	Client receives a "stale file handle" error. The client is using a file handle that is exported to another client address.	Client receives an "access denied" error. The server has already created a list of client addresses that can access the exported file systems.
Server IP address change:	Client is trying to contact old address. If hard mounted then it blocks forever. If soft mounted it timeouts and fails (I/O error).	No errors.
Client network interface down:	Network unreachable error, no matter whether data cached or not at the client.	The mounted directory is empty, i.e., not mounted to anything.
Server network interface down:	If hard mounted client blocks forever. If soft mounted it timeouts and fails (I/O error).	Client cannot mount. Mount process gets "can't get address for server" error.
Client disconnects and reconnects with old IP address:	As long as the reassignment does not take longer than the timeout, the client only experiences a delay.	No errors.
Server disconnects and reconnects with old IP address:	As long as the reassignment does not take longer than the timeout, the client only experiences a delay.	No errors.

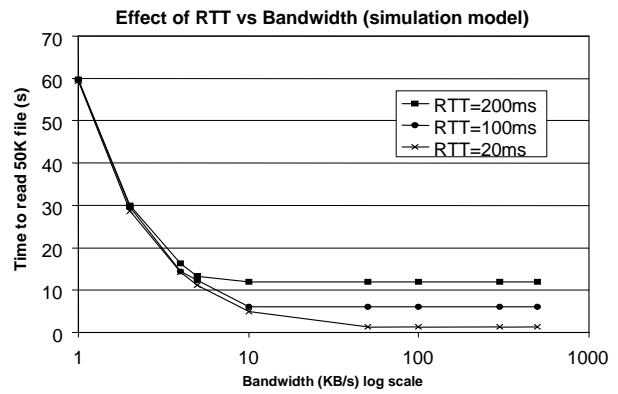
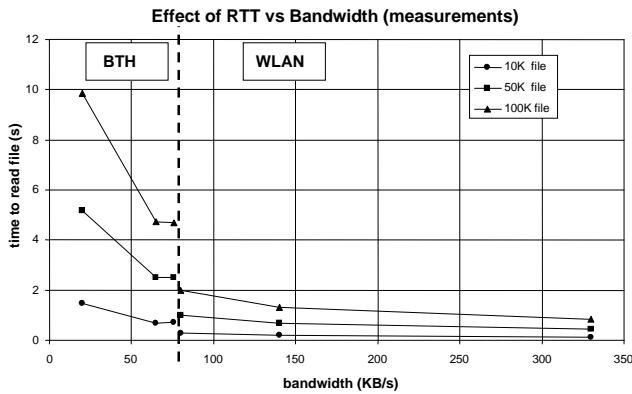


Fig. 2. Effect of bandwidth and RTT.

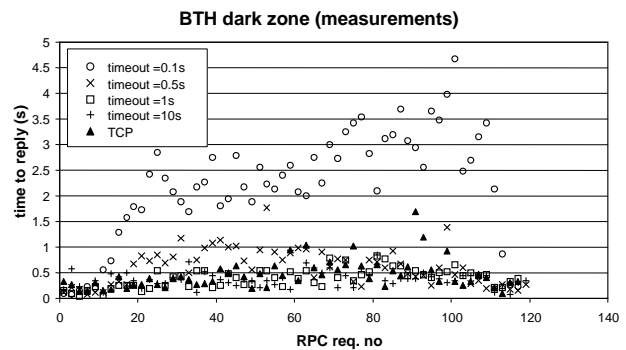
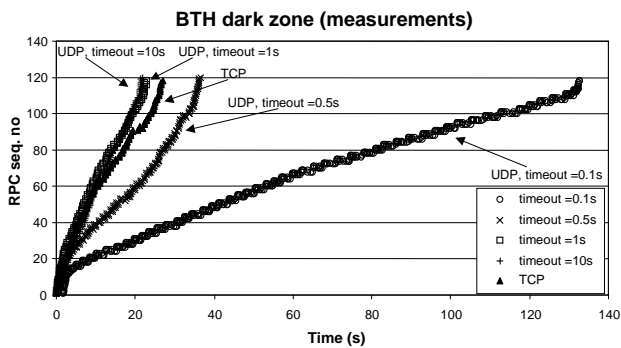


Fig. 3. Effect of high RTT in low quality Bluetooth links.

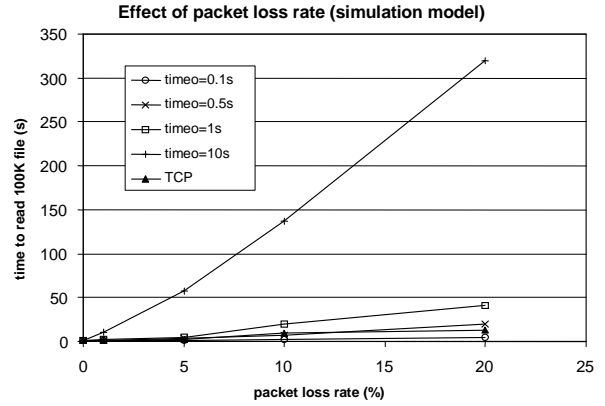
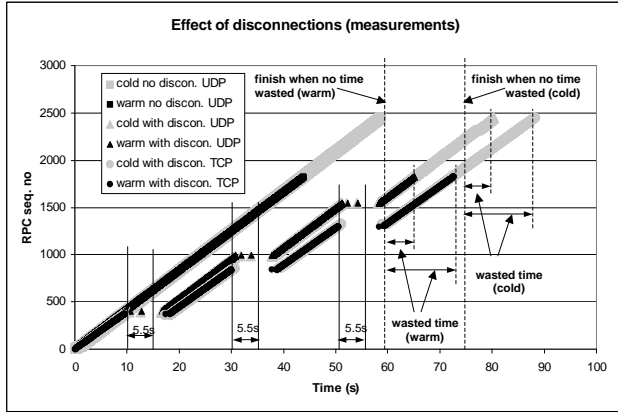


Fig. 4. Effect of temporary disconnections (left) and random packet losses (right).

V. CONSIDERATIONS IN DESIGNING AN M-DFS FOR EPHEMERAL SHARING

In this Section, motivated by the performance results of Section IV, we present some proposals for the design of a RO M-DFS for ephemeral sharing that would operate well in proximity networks. In addition, we describe a possible prototype implementation using the SFS toolkit that we are considering as a future direction.

A. Design Topics and Suggestions

More aggressive caching to support temporary disconnections. Weakly connected networks and frequent disconnections are best countered by the use of aggressive caching. In the case of RO ephemeral file sharing caching mechanisms can be simplified and the state remembered by client and server reduced. That's because no reconciliation mechanisms are necessary to propagate changes back to the server. Callbacks with leases, as described in LBFS [4], offer a rich mechanism that provides cache consistency, while still being able to reduce the amount of client-server communication and provide close-to-open consistency. In addition to file and directory contents, the client should be caching retrieved metadata provided by the server that are interpreted by applications, such as thumbnails or mp3 samples. The amount of data that is to be cached may be controlled by user preferences.

The need for automatic session recovery. Distributed file systems often return errors because of client or server mobility and loss of connectivity. In Section IV-A we have identified two break-down cases. In the first case, both client and server have joined a session (server exported and client mounted the file system) and data may be possibly in transit through the network. A *session support* mechanism would be required to deal with this case. Two issues that need to be considered for session support are deciding whether to continue or drop and restart long pending sessions, and whether to allow applications to know of the disconnections. In the second case, the server has started a session and is waiting for the client to

join (server only exported file system). No data are in transit. A *session recovery* mechanism is required to deal with this case to preserve the file system state related to mounting. Security is an important issue to prevent malicious clients from trying to recover sessions illegally.

Techniques for traffic reduction. To deal with low bandwidth links we suggest traffic reduction techniques. For example Coda uses operation shipping instead of data shipping to reduce bandwidth and LBFS uses zip compression on the data. We suggest using standard data compression techniques as in LBFS because they are simple. Also useful here are application specific data reduction techniques using metadata (e.g. thumbnails, mp3 samples).

Enhanced metadata. The use of metadata can help deal with weak connectivity networks. We distinguish metadata in two types: *application specific* and *file system specific*. Application specific metadata are opaque to the M-DFS. The M-DFS is required to allocate some extra space and possibly an API to applications to be able to use them. File system specific metadata are only seen by the file system and the operating system. These metadata can be used to improve performance, provide enhanced capabilities and make the system more adaptive to the conditions of the mobile environment.

Access control and distributed authentication. Access control lists provide better security than Unix access control as demonstrated in the upcoming NFS v4 protocol [11]. Our suggestion is to use “ephemeral” ACLs, i.e. ACLs that are created and updated by the server owner on the fly to provide the necessary security, while removing stale entries in ACLs and as a result avoiding long ACLs with few active users. Only users that are explicitly given lifelong access to files will not be removed after a certain timeout. With ephemeral file sharing there is an issue of how to provide server and client authentication in a distributed manner without the use of a Certificate Authority. The simplest solution would be to use offline means, such as user to user communication to exchange the necessary verification information.

Naming, discovery and file system removal. An RO M-DFS for ephemeral file sharing does not require a global name space for the remote repositories (e.g. Coda); every server repository can be identified explicitly by the device that exports it. However, using the server name is not mandatory and a discovery mechanism should be flexible enough to follow this and other conventions. In addition, we suggest the use of a uniform name space, so that two clients see the same path to a file from the same server. The service discovery mechanism needs to deal with authentication and privacy. Also, a mechanism is necessary for automatic removal of idle file systems and cached data after a user defined timeout, in order to free device resources.

B. An Implementation Proposal

User level file systems using NFS as the interface to access disks have been used extensively. In particular the SFS toolkit [2] provides an organized set of tools to build such file systems, while avoiding issues with user-level file systems as loop-back servers and clients. In addition, an implementation of the NFS v3 RPC protocol is provided as a skeleton to be extended for adding extra functionality to a file system. The SFS toolkit provides tools to build file systems at user-level with acceptable performance that can be ported to any operating system that supports NFS.

Figure 5 shows how we propose to implement an RO M-DFS using the SFS toolkit. At the server side, a user-level NFS loopback client forwards requests to the local NFS server. At the client side, the NFS client sends requests to a user-level NFS loopback server. A cache manager in the form of a loopback client sends read/write requests to the local NFS server to fetch and store entries in the cache. Having all of the RO M-DFS modules at the user-level facilitates the portability of the filesystem across different platforms.

Currently, we only have a preliminary implementation of an RO M-DFS using the SFS toolkit. At this stage, the implementation includes basic NFS client/server functionality and lease-based caching, similar to the implementation of SFS [3]. Client and server communicate via a customized RPC protocol which is essentially the NFSv3 protocol with minor modifications. Caching is implemented via a cache manager module. Defining the RPC protocol and adding the cache manager module to the client was a straightforward task using the toolkit. Defining the RPC protocol and related data structures required approximately 100 lines of code for the `rpc` compiler of the SFS toolkit. Implementing the prototype cache manager module required approximately 500 lines of C++ code. Our preliminary implementation encouraged us to believe that adding new features to NFS to handle its shortcomings in the mobile environment will not increase the complexity of the filesystem.

VI. CONCLUSIONS AND FUTURE DIRECTIONS

We have studied the performance of NFS in the mobile environment and have identified several shortcomings of the protocol when used under difficult network conditions such as

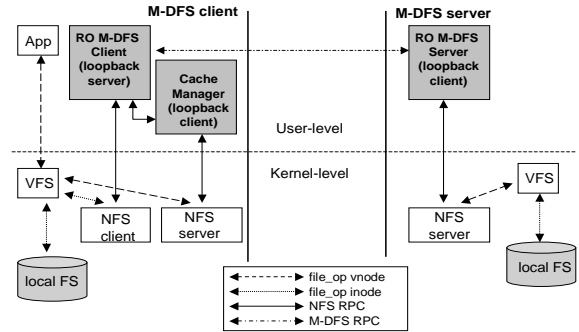


Fig. 5. M-DFS implementation based on the SFS toolkit [2].

disconnections, packet losses and low signal reception zones, which are typical in proximity networks. Based on our observations and the application requirements for ephemeral file sharing, we have suggested several enhancements necessary in order to design an RO M-DFS suitable for ephemeral sharing.

Although, the suggested enhancements have been validated by previous systems, a fully-fledged RO M-DFS implementation is necessary to validate them as a working whole. Our current implementation is incomplete and only suggests that adding new features will not come at high complexity cost. In the future, we plan to implement our design suggestions and evaluate a fully functional RO M-DFS using the same experimental setup and simulation infrastructure we used to evaluate standard NFS performance.

We have chosen to base an RO M-DFS on NFS, because of its ubiquitous use, simple design and well-understood implementation compared to other networking file systems. This makes NFS suitable for ephemeral file sharing. An alternative, that is currently under consideration, is an HTTP-based approach for ephemeral file sharing. HTTP is attractive because, like NFS, it is stateless and ubiquitous. File sharing over HTTP can be implemented by having server and client proxies that handle caching and content transformation to reduce bandwidth requirements. A disadvantage, however, compared to a filesystem-based approach is the lack of a common file access interface for applications. In the future, we are planning to evaluate the two alternatives using our experimental setup and simulation infrastructure.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Andre Dolenc of Nokia for the motivation, discussions and funding he provided for this work. We also thank the reviewers of the workshop for their valuable comments and feedback.

REFERENCES

- [1] B. Callaghan, B. Pawlowski, and P. Staubach. NFS version 3 Protocol Specification, RFC 1813, June 1995.
- [2] D. Mazières. A toolkit for user-level file systems. In *USENIX Technical Conference, Boston, MA*, June 2001.
- [3] D. Mazières, M. Kaminsky, M. Frans Kaashoek, and E. Witchel. Separating key management from file system security. In *Proceedings of the 17th ACM Symp. on OS Principles*, December 1999.

- [4] A. Muthitacharoen, B. Chen, and D. Mazières. A Low-bandwidth Network File System. In *Symposium on Operating Systems Principles*, pages 174–187, 2001.
- [5] J. J. Kistler and M. Satyanarayanan. Disconnected Operation in the Coda File System. In *Thirteenth ACM Symp. on OS Principles*, 1991.
- [6] P. J. Braam and P. Nelson. Removing Bottlenecks in Distributed Filesystems: Coda and Intermezzo as examples. In *5th Annual Linux Expo*, May 1999.
- [7] Cynthia D. Rais and Satish K. Tripathi. Measuring NFS Performance in Wireless Networks. Technical Report CS-TR-3582, 1995.
- [8] Rohit Dube, Cynthia D. Rais, and Satish K. Tripathi. Improving NFS Performance Over Wireless Links. *IEEE Trans. on Comp.*, 46(3):290–298, 1997.
- [9] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. Frans Kaashoek. The Click Modular Router. *ACM Trans. Comput. Syst.*, 18(3):263–297, 2000.
- [10] John C. S. Lui, Oldfield K. Y. So, and T. S. Tam. NFS/M: An Open Platform Mobile File System. In *Int. Conf. on Distr. Comp. Syst.*, pages 488–495, 1998.
- [11] Shepler S. et al. Network File System version 4 Protocol, RFC 3530, April 2003.