MobileFS: Mobile SSHFS done right

Mainak Chowdhury, Alon Kipnis
CS344G Final Report

March 13, 2015

Abstract

MobileFS aims to be an enhanced SSHFS implementation, designed specifically for networks with intermittent connectivity and high latency. It provides a userspace program for mounting directories on remote servers. One of the main differences from SSHFS is in its prioritization of short system calls (as compared to calls which may take a long time to complete), thereby enhancing responsiveness. Moreover it is perfectly stateless, and exposes full control of pending operations to the user. This helps prevent system calls to the local mountpoint from freezing the entire system due to roaming/loss of network connectivity. MobileFS is implemented in Python and available at https://github.com/mainakch/mobilefs.

1 Introduction

Network filesystems provide an abstraction of a local filesystem over a network. They allow the client to read and write a file on a remote computer as if the file were on the user's local filesystem. While the use of network filesystems has increased especially with faster and faster networks, setting up a network filesystem still requires considerable setup and maintenance effort and is not feasible for personal use or for small scale networks. This is where userspace filesystems come in. With a setup and maintenance complexity no greater than that required to setup remote shell (e.g. SSH) access, with **userspace filesystems**, one can have most of the benefits of a network filesystem. SSHFS [1] is such an userspace filesystem which requires only an SFTP server on the server and FUSE [3](userspace filesystem) on the client side.

SSHFS is built on top of a remote shell session. While this enables a simple setup, the persistency of an SSH session leads to a poor user experience, especially with unreliable networks or with roaming. Paradoxically, even with the proliferation of fast networks, our reliance on unreliable connections such as LTE and WiFi has increased as well. This unreliability leads to unacceptable delays and unexpected system freezes with SSHFS.

MobileFS is an userspace filesystem designed to address these limitations.

2 Where SSHFS fails

In this section we will describe the basic architecture of SSHFS and the kind of behavior that this architecture leads to in unreliable networks.

When one mounts a remote directory to a local mountpoint using SSHFS, the client side of SSHFS listens to a local mountpoint using FUSE. Every system call in this mountpoint is directed by FUSE to SSHFS. SSHFS translates that system call into an SFTP command (or a sequence of SFTP commands) and sends it to the remote shell (via SSH). SSHFS returns the system call only after the operation is reliably transmitted and a response, if needed, is received from the server side.

As a result of this architecture, SSHFS suffers from the following:

- **Latency:** Filesystem operations such as reading or writing a large amount of data may take a long time to complete over a slow network connection. In the meantime, operations with less overhead (such as working with a text editor or navigation commands) experience unacceptable latencies. This is because these operations are required to wait 'in line' until the system calls issued before have returned.

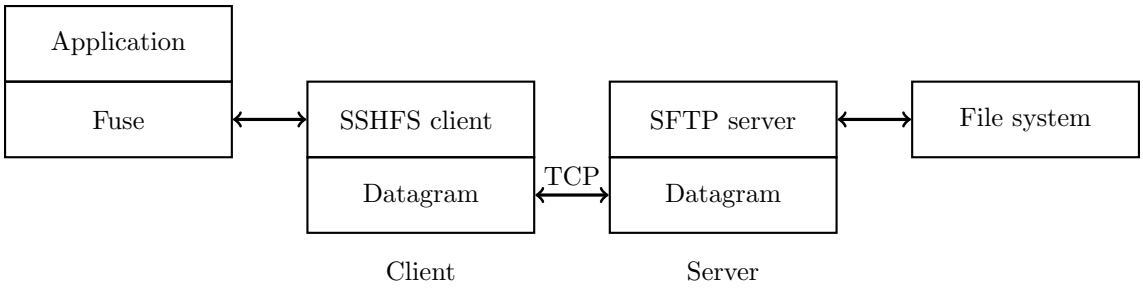

Figure 1: Schematic of SSHFS

- **Network failure recovery:** SSH sets up a remote shell session and as such is inherently susceptible to network disconnects and to roaming. In particular, network failure due to disconnects or roaming among networks interrupts an SSH session which may end up freezing the application that issued the original system call. This freeze can usually be resolved only by killing the application process, even if the network connection has been reestablished (especially after roaming).

3 MobileFS

Towards addressing the frustrations described above, we present MobileFS: an userspace network filesystem. Before describing the architecture we present the workflow involved in using it, together with the program design that enables the workflow.

3.1 Using MobileFS

MobileFS requires a server process to be started on the remote machine:

```
corn29:~/mobilefs> python network_server.py corn29.stanford.edu 60002
```

On the client machine we issue

```
corn29:~/mobilefs> python curses_gui.py corn29.stanford.edu 60002 /remote_mnt_path ~/local_mnt_path
```

Here `60002` is the UDP port that the server listens to. This command mounts the remote directory `/remote_mnt_path` on `corn29.stanford.edu` to the local mount point `~/local_mnt_path`. It also starts up an interface where the list of pending system calls is displayed. In case a system call takes long to return, it can be killed by the user. One can unmount the remote filesystem through this interface.

3.2 MobileFS protocol design goals

In this section, we list some considerations that went into the design for MobileFS. In particular, we designed MobileFS to:

- Prioritize asynchronous small requests over large (or long running) requests to enhance responsiveness
- Emphasize transparency of the underlying filesystem operations (with user intervention and non-blocking operations whenever possible)

We next present an architecture towards implementing the above goals (Figure 3). The main architectural change over SSHFS is the priority queue on the client side together with a datagram protocol. In this queue, large write/read commands will be given lower priorities over other operations which require less network bandwidth to complete. In addition, we expose the state of that queue to the user and allow interventions such as killing a specific operation.

We list some features supporting the above operations in the following.

Figure 2: Operations are prioritized by their size. The 'kill' command removes the operation from the transmission queue and returns with error (EINTR error code).

3.3 MobileFS features

- **Custom datagram protocol instead of a persistant connection**: We implement a custom datagram protocol to handle out-of-order filesystem requests reliably and quickly. Components of this implementation include logic for prioritizing "short" or "important" system calls and a congestion control mechanism.
- **Stateless**: Our implementation does not attempt to maintain any state (of pending requests for example). Instead it focuses on transporting datagrams to the remote end on a best effort basis, retransmitting as necessary if there is a timeout. We do not aim to resolve conflicts or situations where a "short" request may interfere with a large request (e.g. deleting a file while a large write operation to the same file is in progress).
- **User control**: We expose the state of pending and completed system calls to the user. This lets the user cancel system calls in progress if they end up taking long (due to an unresponsive network for example).
- **Congestion control**: Our current implementation is based on an additive increase and multiplicative decrease protocol for the sender window, with a slow start phase. The congestion signals are the packet timeout events.

3.4 Future work/known issues

In this section we list some known issues and potential improvements to our implementation.

- **Security**: The current implementation does not yet have security built-in. We could however use the same ideas for encrypting datagrams as were considered in the implementation of the Mobile Shell [4]. In particular, we could use the symmetric key based AES-128 in Offset Codebook Mode. The packet headers could be used as nonces. Alternatively, one could use DTLS [2].
- **Packet overhead**: We did not try to optimize the datagram packet header, nor the size of the datagrams to adjust to network capabilities. Any savings in the header information will translate to an increase in the achievable throughput.

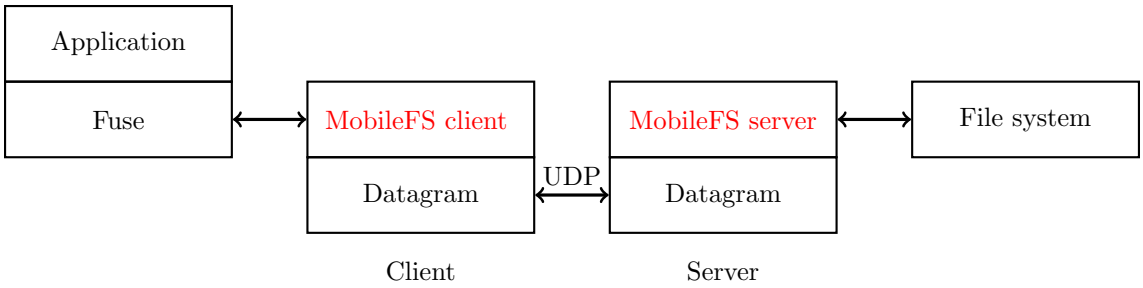

Figure 3: Schematic of MobileFS (our modifications in red)

- **Better congestion control**: While we implemented a basic congestion control, the throughput delay performance can be significantly improved by taking into account round trip time estimates and explicit congestion notifications where available.
- **Better handling of suspend and resume**: In our tests, FUSE prevented the system from suspending if there were in-progress system calls. This could potentially be handled in MobileFS by raising an error on catching a SIGSTOP and returning from the system call.

4 Results: MobileFS vs SSHFS

A comparison between MobileFS and SSHFS in various scenarios is summarized in the following table:

Scenario	SSHFS	MobileFS
Executing a short read and write command to/from the remote folder while a large file is being written to it	9 seconds on average	0.85 seconds on average
Network disconnect	returns after a timeout predetermined by ServerAliveInterval	returns after user kills all the pending commands
Roaming between wireless networks while a writing command is in progress	does not reconnect and returns an error after a timeout	reconnects and completes the writing operation

5 Conclusion

MobileFS is an userspace filesystem designed with the mobile user in mind. Compared to existing implementations (SSHFS) our implementation allows us to guarantee **lower latencies**, **transparency of pending system calls** and **robustness to roaming** or **intermittent connectivity** in networks. We do this by implementing a prioritization scheme for system calls on top of the unreliable datagram layer. This ensures that system calls with a large network overhead do not end up blocking system calls which may require a much lower overhead. In addition, our implementation is robust to roaming or loss of network connectivity.

6 Acknowledgements

We would like to thank Keith Winstein for proposing this project, and for being an awesome instructor for CS344G.

References

[1] Hoskins, M. E. SSHFS: super easy file access over SSH. *Linux Journal 2006*, 146 (2006), 4.

[2] Modadugu, N., and Rescorla, E. The design and implementation of datagram TLS. In *NDSS* (2004).

[3] Szeredi, M. Fuse: Filesystem in userspace. 2005. *URL http://fuse. sourceforge. net*.

[4] Winstein, K., and Balakrishnan, H. Mosh: An interactive remote shell for mobile clients. In *USENIX Annual Technical Conference* (2012), pp. 177–182.