Mosh: An Interactive Remote Shell for Mobile Clients

Keith Winstein and Hari Balakrishnan
M.I.T. Computer Science and Artificial Intelligence Laboratory, Cambridge, Mass.
{keithw,hari}@mit.edu

Abstract

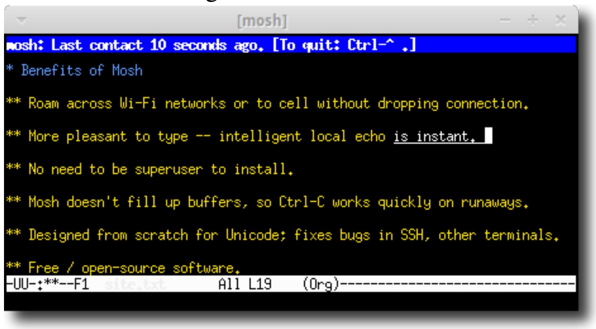
Mosh (mobile shell) is a remote terminal application that supports intermittent connectivity, allows roaming, and speculatively and safely echoes user keystrokes for better interactive response over high-latency paths. Mosh is built on the State Synchronization Protocol (SSP), a new UDP-based protocol that securely synchronizes client and server state, even across changes of the client's IP address. Mosh uses SSP to synchronize a character-cell terminal emulator, maintaining terminal state at both client and server to predictively echo keystrokes. Our evaluation analyzed keystroke traces from six different users covering a period of 40 hours of real-world usage. Mosh was able to immediately display the effects of 70% of the user keystrokes. Over a commercial EV-DO (3G) network, median keystroke response latency with Mosh was less than 5 ms, compared with 503 ms for SSH. Mosh is free software, available from `http://mosh.mit.edu`. It was downloaded more than 15,000 times in the first week of its release.

1 Introduction

Remote terminal applications are almost as old as packet-switched data networks. The most popular such application today is the Secure Shell (SSH) [9], which runs inside a terminal emulator. Unfortunately, SSH has two major weaknesses that make it unsuitable for mobile use. First, because it runs over TCP, SSH does not support roaming among IP addresses, or cope with intermittent connectivity while data is pending, and is almost unusable over marginal paths with non-trivial packet loss. Second, SSH operates strictly in character-at-a-time mode, with all echoes and line editing performed by the remote host. On today's commercial EV-DO and UMTS (3G) mobile networks, round-trip latency is typically in the hundreds of milliseconds when unloaded, and on both 3G and LTE networks, delays reach several seconds when buffers are filled by a concurrent bulk transfer. Such delays often make SSH painful for interactive use on mobile devices.

This paper describes a solution to both problems. We have built **Mosh**, the mobile shell, a remote terminal application that supports IP roaming, intermittent connectivity, and marginal network connections. Mosh performs predictive client-side echoing and line editing without any change to server software, and without regard to which application is running. Mosh makes remote servers feel more like the local computer, because most keystrokes are reflected immediately on the user's display—even in full-screen programs like a text editor or mail reader.

Figure 1: Mosh in use.



```
mosh: Last contact 10 seconds ago. [To quit: Ctrl-^ .]
* Benefits of Mosh

** Roam across Wi-Fi networks or to cell without dropping connection.

** More pleasant to type -- intelligent local echo is instant.

** No need to be superuser to install.

** Mosh doesn't fill up buffers, so Ctrl-C works quickly on runaways.

** Designed from scratch for Unicode; fixes bugs in SSH, other terminals.

** Free / open-source software.
-UU-:**--F1  site.txt       All L19    (Org)-------------------------------
```

These features are possible because Mosh operates at a different layer from SSH. While SSH securely conveys an octet-stream over the network and then hands it off to a separate client-side terminal emulator to be interpreted and rendered in cells on the screen, Mosh contains a server-side terminal emulator and uses a new protocol to synchronize terminal screen states over the network, using the principle of application-layer framing [3].

Because both the server and client maintain an image of the screen state, Mosh can support intermittent connectivity and local editing, and can adjust its network traffic to avoid filling network buffers on slow links. As a result, unlike in SSH, in Mosh "Control-C" always works to cease output from a runaway process within an RTT.

Mosh's design makes two principal contributions:

1. **State Synchronization Protocol:** A new secure object synchronization protocol on top of UDP to synchronize abstract state objects in the presence of roaming, intermittent connectivity, and marginal networks (§2).
2. **Speculation:** Mosh maintains the screen state at both the server and client and uses the above protocol to synchronize them (§3). The client makes guesses about the effect each new keystroke will have on the screen, and when confident renders the effects immediately. The client verifies its predictions and can repair the screen state if necessary.

We have implemented Mosh in C++ and have experimented across various networks and across disconnec-