Mosh
An Interactive Remote Shell for Mobile Clients

Keith Winstein (with Hari Balakrishnan)

keithw@mit.edu

May 16, 2012

Secure Shell, 1996

- Connects local terminal to remote terminal.
- Conveys over TCP:
  - user keystrokes $\Rightarrow$ server
  - octet stream (coded screen updates) $\Rightarrow$ client terminal
- Connection endpoints dictated by IP:port on both sides

Post-1996 problems with SSH

- Can't roam:
  - . . . across Wi-Fi networks.
  - . . . from Wi-Fi to cell or vice versa.
- TCP times out if data unacknowledged after n minutes.
  - Session dies if laptop goes to sleep.
- TCP responds poorly to packet loss.

More problems with SSH

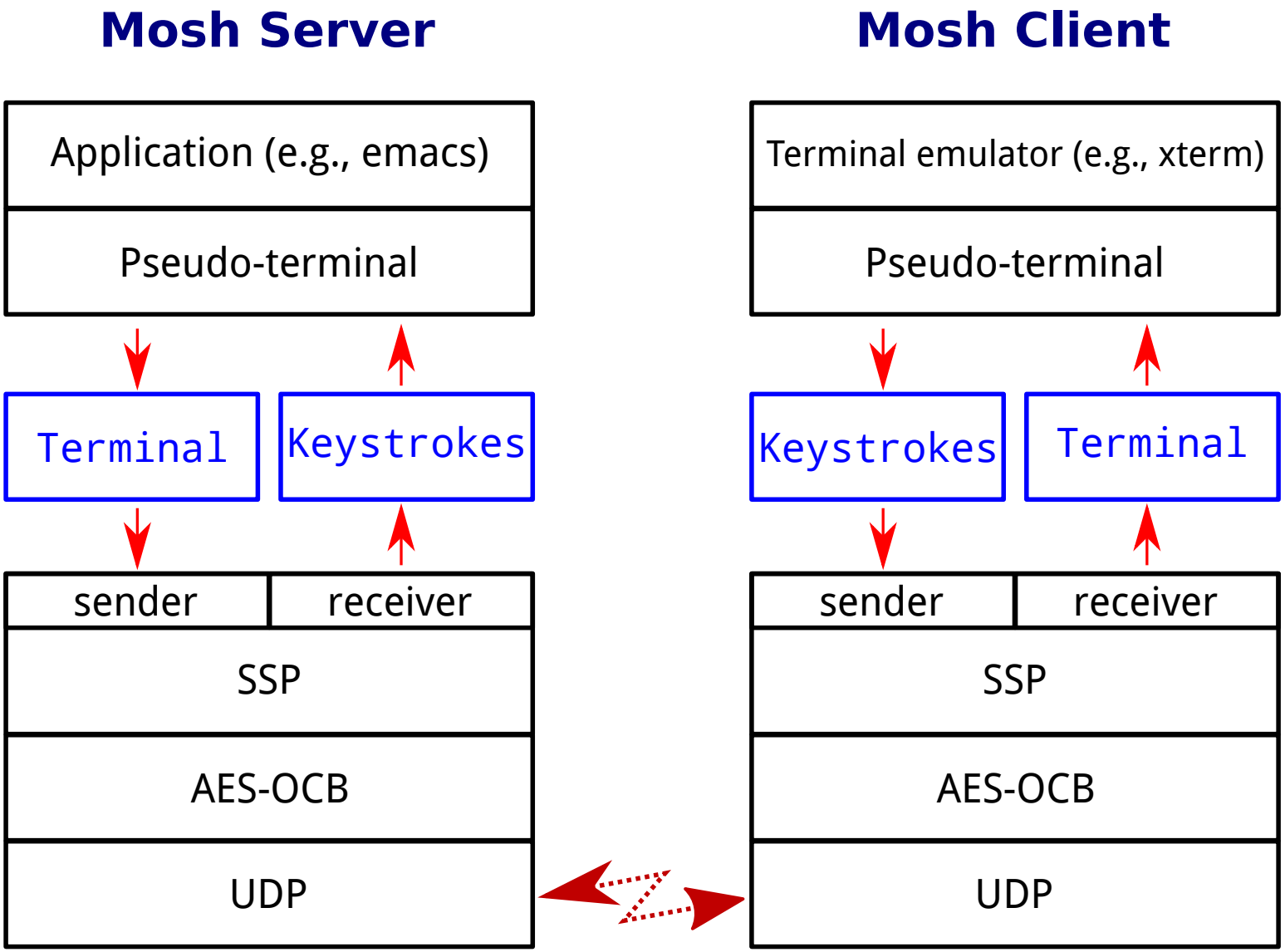
- Byte stream is wrong layer of abstraction for screen.
  - If client screen state is old, want to skip ahead directly.
  - Don't want to replay megabytes in between.
  - But SSH doesn't know how client interprets octets, so must send all.
  - TCP fills buffers, so bufferbloat means Control-C takes forever.
- Typing and editing on high-latency path is frustrating.
  - Cellular wireless (100 ms to 500 ms)
  - Intercontinental (250 ms)
  - Loaded 4G LTE (5,000 ms!)

Solution 1: State Synchronization Protocol

- Runs over UDP.
- Instead of synchronizing *octet streams*, synchronize *objects*.
- Object represents state of endpoint.
- Implements simple interface:
  - `diff`: make vector from state $A \rightarrow B$
  - `patch`: apply vector to A, producing B
  - `subtract`: remove common prefix from A, B
- Object implementation, **not protocol**, defines synchronization semantics.

State Synchronization Protocol (cont.)

- Runs over UDP
- Protected by AES-OCB (Krovetz 2011)
  - Integrity and confidentiality with one key.
- Roaming is easy:
  - Source address of latest authentic datagram from client $\Rightarrow$ new destination address for server.


Mosh Server
Mosh Client
Application (e.g., emacs)
Pseudo-terminal
Terminal
Keystrokes
sender
receiver
SSP
AES-OCB
UDP
Terminal emulator (e.g., xterm)
Pseudo-terminal
Keystrokes
Terminal
sender
receiver
SSP
AES-OCB
UDP

State Synchronization Protocol (cont.)

- **Flow control**: try to have at-most one `diff` in flight.
- Use TCP's SRTT/retransmission timeout with tweaks.
- Minimum interval between frames is SRTT/2.
- Can skip intermediate states.
- "Pretransmissions" (post-paper improvement).

Pretransmissions (post-paper improvement)

"Prophylactic" retransmission reduces latency in presence of loss.

1. Last ack was for state #3. Then state changes to #4.
2. Host sends `diff` from $3 \rightarrow 4$.
3. Object state changes to state #5.
4. If last `diff` hasn't timed out, formulate next `diff` as $4 \rightarrow 5$.
5. **Also** make `diff` from $3 \rightarrow 5$: the *pretransmission*.
6. If retransmission is shorter or not "much" longer, send it instead.

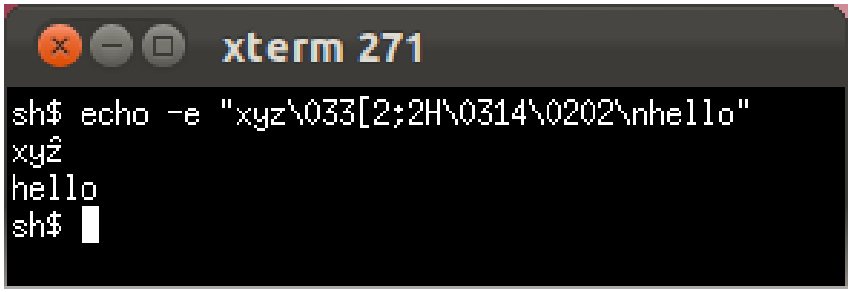
Solution 2: Different semantics for different directions

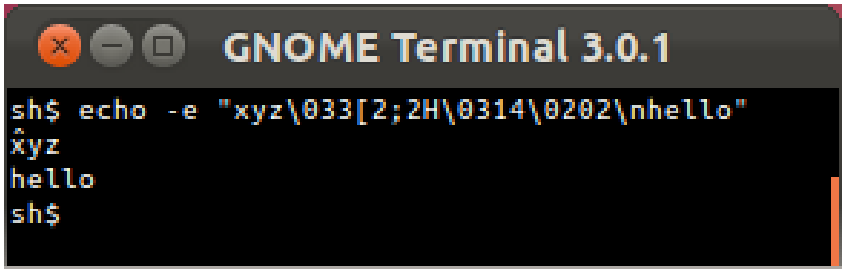
- Client ⇒ server: object represents *history of keystrokes typed*.
  - Same semantics as TCP.
  - `subtract` ensures only outstanding data actually stored in memory.
- Server ⇒ client: object represents *current screen state*.
  - *E.g.* 80x24 terminal, Unicode grapheme in each cell + bell.
  - Okay to skip over intermediate states!
  - Server and client **both have model** of what's on the screen.

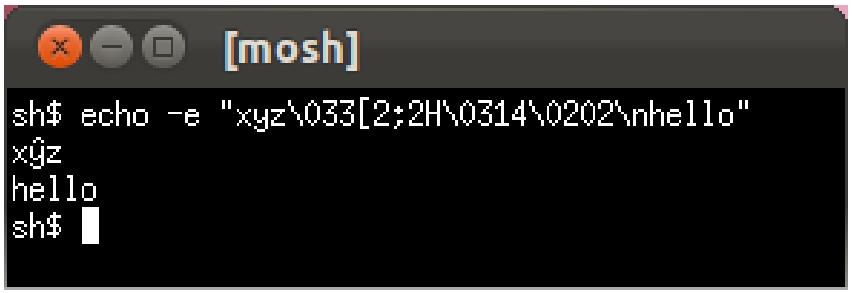
Benefits

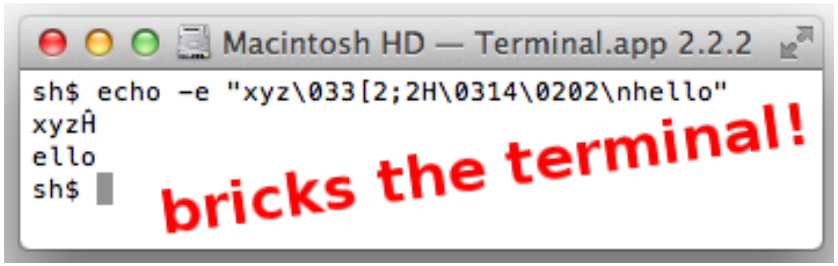
- Can sleep and wake up.
- Can roam across ESSIDs or to cell network.
- Can fly to Canada and open laptop in hotel.
- Can warn user when displayed state is stale.
- Semantically appropriate flow control won't fill up network queues.
  - Control-C always works within RTT.
  - No beeping fits (one beep per `diff`).
- Supports lossy links.
- Uses SSH to start connection.
  - No privileged (root) code!
  - No daemons.
- Better Unicode support.

Unicode admits varying interpretations.









Solution 3: Speculative Local Echo and Editing

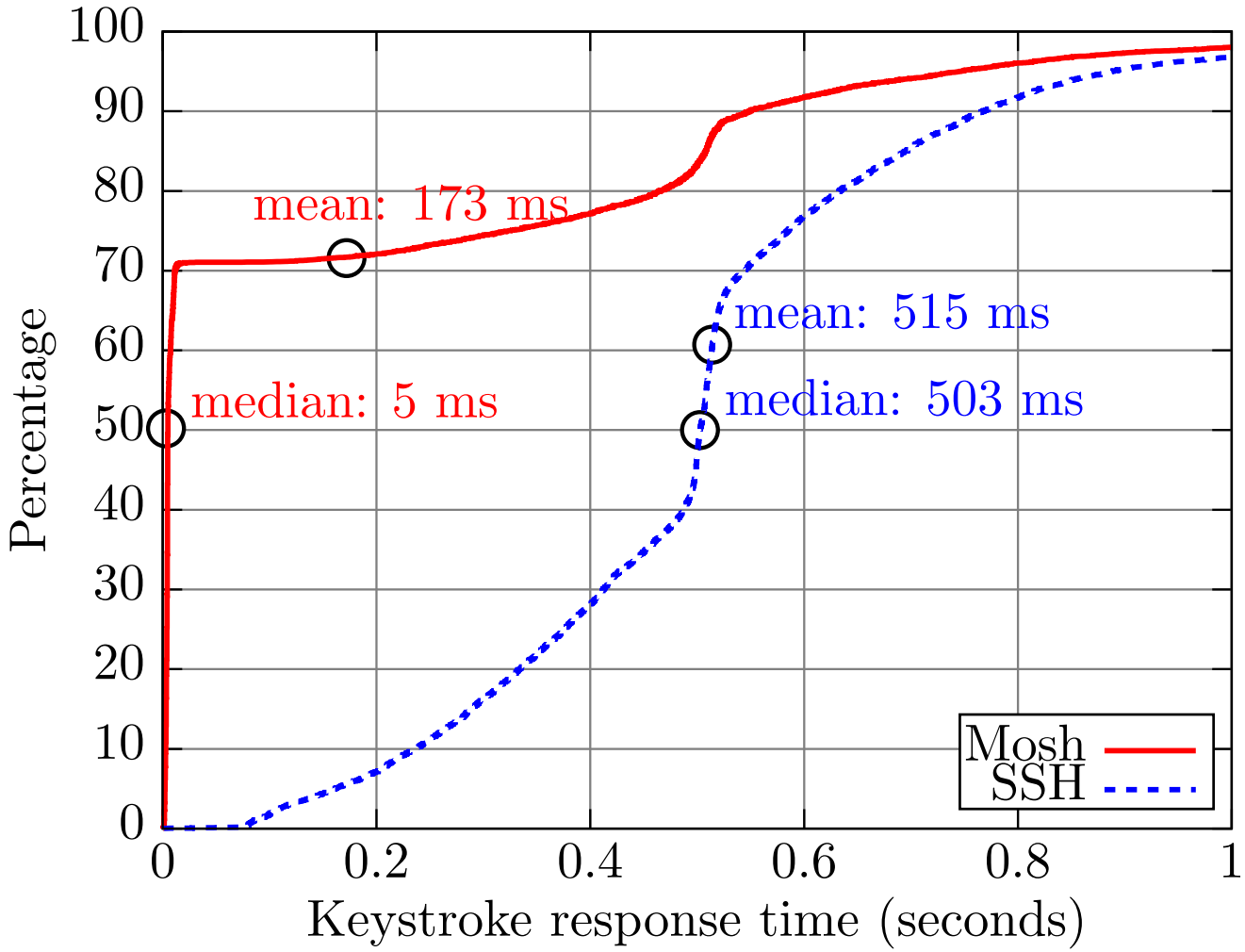
- Now that client has complete picture of screen state, can start to anticipate server response.
- Runs predictive model in the background.
  - If user hits keystroke, predict key will appear where cursor was.
- Make predictions in *epochs*.
- If any prediction from epoch n is confirmed, show all predictions made in that epoch.
  - n might be current epoch or one from the past.
- If user does something difficult to handle, become *tentative* by incrementing epoch.
  - Moves to new row.
  - Hits control character (including carriage return).
  - Uses up/down arrow key.

Demo

Evaluation

- Collected 40 hours of terminal usage from six anonymous users.
- Covers 9,986 keystrokes using shell, e-mail, text editor (emacs and vi), chat, Web browser.
- Replayed sessions over Sprint 1xEV-DO (3G) commercial network over SSH and Mosh.
- Result: 70% of keystrokes can be predicted instantly.
- Prediction errors $< 1\%$
  - Most common cause: word wrap
  - Bad prediction removed from display within RTT.
- Median UI latency **503 ms** (SSH) to $<$ **5 ms** (Mosh).
- Mean UI latency **515 ms** (SSH) to **173 ms** (Mosh).

Keystroke response times, cumulative distribution



Evaluation (cont.)

Verizon LTE service in Cambridge, Mass., running one concurrent TCP download:

	Median latency	Mean	σ
SSH	5.36 s	5.03 s	2.14 s
Mosh	$<$ 0.005 s	1.70 s	2.60 s

MIT-Singapore Internet path (to Amazon EC2 data center):

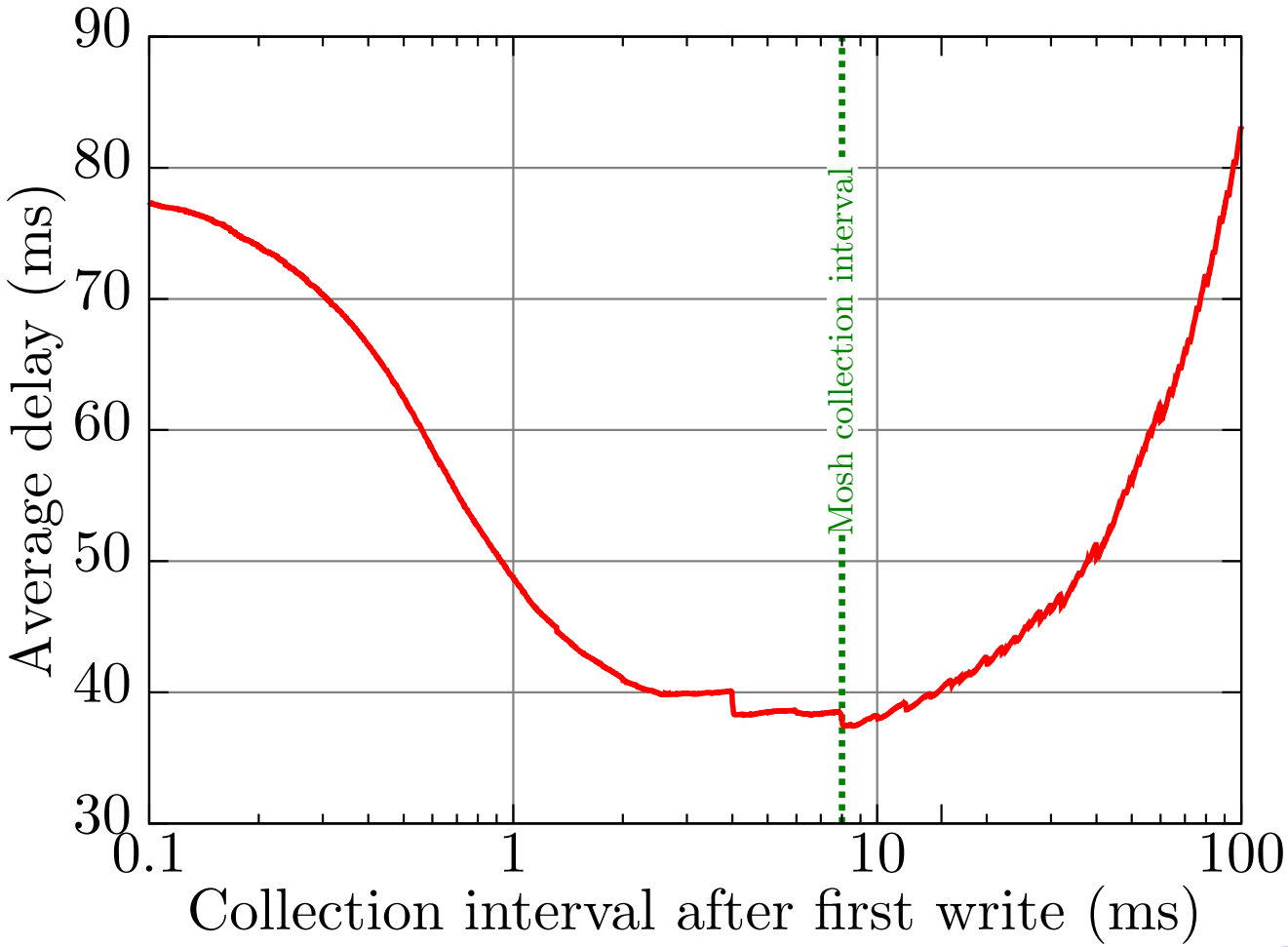
	Median latency	Mean	σ
SSH	273 ms	272 ms	9 ms
Mosh	$<$ 5 ms	86 ms	132 ms

SSP with high packet loss

Synthetic link with 100 ms RTT, 50% round-trip i.i.d. packet loss:

	Median	Mean	σ
SSH	0.416 s	16.8 s	52.2 s
Mosh (no predictions)	0.222 s	0.329 s	1.63 s

Average protocol-induced delay from varying collection interval (SRTT $>$ 500 ms)



Deployment

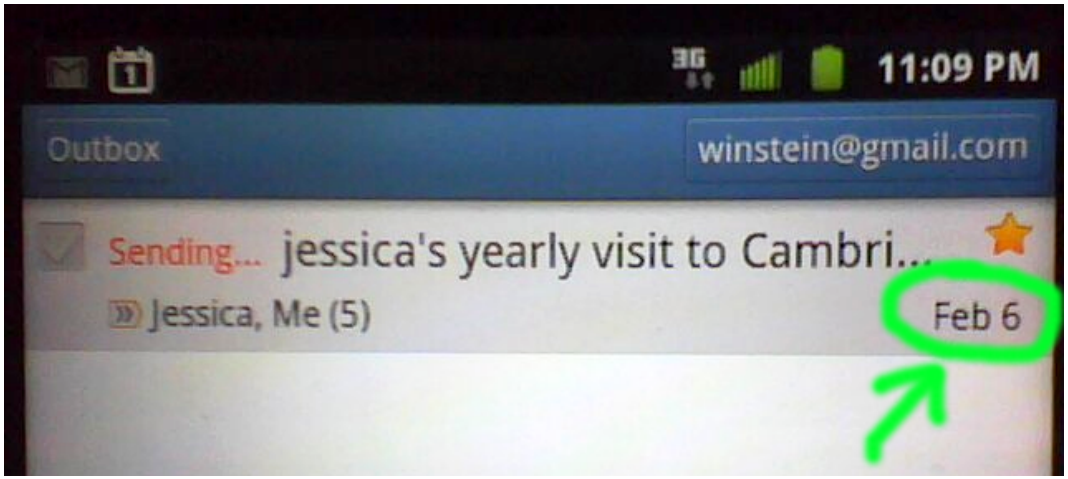
- Distributed in Debian, Ubuntu, Fedora versions of GNU/Linux.
- Included in MacPorts, Homebrew, FreeBSD ports collections.
- Web site at `mosh.mit.edu`
- News stories in April on Hacker News, Reddit, The Register, Twitter, Slashdot, Barrapunto.
- 130,000 unique visitors, 25,000+ downloads, 1,100+ "followers" of source code repository on Github.

Reception

- "one of those times you don't realize something is broken until you see it fixed" — @xlfe
- "If you are an SSH user, check out mosh.mit.edu - the user experience really is dreamy." — @adamhjk
- "mosh is awesome. Tested it for two weeks and it really made my life easier: faster feedback and no more reconnects(!)" — @esmolanka
- "Finalement, la vie d'admin c'est pas si Mosh que ca" — @korben
- "There is very (if any) little research content." — USENIX review
- "ISO 2022 locking escape sequences oh flying spaghetti monster please kill me now." — USENIX review

State Sync Protocol for all?

- We believe SSP may be appropriate for many network problems.
- Android Gmail, Google Chat cannot roam without failure.
- **May 15, 2012**:



- Neither can Gmail (Web edition).
- These problems can be expressed as state synchronization.

Next Steps

- Mosh paper to be presented at USENIX ATC (June 2012).
- Mosh software under development by a team of contributors.
- We are working to apply SSP to mobile videoconferencing.
- We hope to show quantitative improvement on standard metrics (latency, quality), plus features like roaming.

Summary

- SSP is a secure datagram protocol that synchronizes abstract objects across a roaming IP connection.
- Mosh uses SSP to synchronize a terminal emulator with predictive local echo.
- In evaluations with 10,000 real-world keystrokes from six users, Mosh markedly reduced user-visible latency across several Internet paths.
- We think SSP will be useful for other applications as well.
- `http://mosh.mit.edu`