Capture Protocol Timing Feedback

Empirical Analysis and Proposal for ext-image-copy-capture-v1

Author: Greg Lamberson, **Lamco** Development
Date: March 29, 2026
Context: wayland-protocols issue #309
Data source: lamco-probe diagnostic suite (ext-image-copy-capture-v1 + pw-profiler)
Compositors tested: Sway 1.11 (2 distros), Hyprland 0.54.2, Mutter 49

1. Executive Summary

This report presents empirical evidence that capture clients using ext-image-copy-capture-v1 cannot make informed encoding decisions because the `request→ready` interval is opaque. The data demonstrates:

1. **Orders-of-magnitude variance:** The same capture session on a single compositor produces request→ready intervals ranging from 6ms to 10,001ms — an 1,800× range.
2. **Cross-compositor divergence:** Under identical interactive desktop use, Hyprland delivers 1,953 frames/90s (21.7 FPS, p50=44ms) while Sway delivers 315 frames/90s (3.5 FPS, p50=164ms). Both implement ext-image-copy-capture-v1.
3. **Decomposition impossible:** A client cannot determine whether a 164ms request→ready interval was 160ms of compositor work or 160ms of idle waiting. The encoding decision should differ dramatically between these cases.
4. **Compositors have the data:** The protocol requires `ready` to fire after the buffer copy completes. The compositor knows when the copy started and finished — this timing exists but is not exposed.

2. Measurement Methodology

2.1 Tooling

All measurements collected with **lamco-probe**, a Wayland remote desktop stack diagnostic tool. The capture probe connects as a native Wayland client, creates an ext-image-copy-capture-v1 session, allocates SHM buffers, and runs a timed capture loop measuring:

- **request→ready:** CLOCK_MONOTONIC interval from `capture` request to `ready` event
- **Damage:** rectangle count and pixel area per frame
- **Presentation time:** compositor-reported display timestamp

The probe does NOT process pixel data or encode frames — it measures protocol-level timing only.

2.2 Environment

All tests on Proxmox VE QEMU/KVM virtual machines with virtio-vga display. Buffer: SHM Xrgb8888, resolution 1280×800.

VM	Compositor	Version	Infrastructure	OS
fedora-sway	Sway	1.11	wlroots 0.19	Fedora 43
endeavouros	Sway	1.11	wlroots 0.19	EndeavourOS
archie	Hyprland	0.54.2	Aquamarine 0.10	Arch Linux

3. Results

3.1 Idle Desktop — The Decomposition Problem

60-second captures, no user interaction:

Compositor	VM	Frames	FPS	avg	min	max	p50	p95
Sway 1.11	Fedora	8	0.1	6,886ms	13ms	9,999ms	9,991ms	9,999ms
Sway 1.11	EndeavourOS	61	1.0	979ms	16ms	1,024ms	1,002ms	1,015ms
Hyprland 0.54.2	Arch	1	—	—	—	—	—	—

> **Key Finding: The Decomposition Problem**
>
> A client receiving a frame with 1,002ms request→ready on EndeavourOS Sway cannot know that this represents "1,000ms waiting for the clock to tick, then 2ms copying the changed region." It looks identical to a hypothetical scenario where the compositor spent 1,000ms on an expensive GPU operation.

Fedora Sway produces 8 frames in 60 seconds — its first frame arrives in 13ms (actual compositor work), but subsequent frames arrive at ~10s intervals (probe timeout, no screen changes). EndeavourOS Sway shows a steady 1Hz cadence from periodic status bar updates. The same Sway version, different distro, different behavior.

3.2 Interactive Desktop — Cross-Compositor Divergence

90-second captures with real user activity (applications open, typing, moving windows):

Compositor	Frames	FPS	avg	min	max	p50	p95	Damage
Sway 1.11	315	3.5	286ms	6ms	1,033ms	164ms	1,005ms	26K px
Hyprland 0.54.2	**1,953**	**21.7**	**46ms**	**19ms**	**148ms**	**44ms**	**83ms**	**1,024K px**

> **Key Finding: 6× Divergence on the Same Protocol**
>
> Hyprland delivers **6.2× more frames** with **3.7× lower median latency** than Sway under comparable interactive use. Both compositors implement ext-image-copy-capture-v1. The difference is entirely in damage tracking and frame scheduling policy — invisible to the client.

A VNC/RDP server on Sway has a highly variable frame budget — sometimes 6ms per frame, sometimes 1,033ms. On Hyprland, the budget is consistently 20–80ms. These two compositors require fundamentally different encoding strategies, but the client cannot determine which regime it's in.

3.3 Idle vs Interactive — Same Compositor, Different Behavior

Scenario	Frames/90s	FPS	avg	p50	min	max
Idle	~90	1.0	979ms	1,002ms	16ms	1,024ms
Interactive	**315**	**3.5**	**286ms**	**164ms**	**6ms**	**1,033ms**

The same compositor, same configuration — 3.5× more frames when the user is active. But a client seeing a 164ms frame cannot distinguish Case A (user clicked, 4ms copy, 160ms was queue wait) from Case B (complex render, 160ms copy, 0ms wait).

3.4 PipeWire Path — Mutter Comparison

For compositors that use PipeWire ScreenCast (GNOME/Mutter, KDE/KWin), `pw-profiler` data shows frame delivery is **consumer-dependent**:

Consumer	FPS	avg interval	max gap	Process time	Xruns
gnome-remote-desktop	32.7	30.5ms	443.7ms	24µs	0
lamco-rdp-server	17.2	58.3ms	199.3ms	1,794µs	112

The compositor's scheduling changes based on the consumer's processing speed — another dimension of the opacity problem.

4. The Protocol Gap

4.1 What the Client Cannot Compute

- **Copy duration:** The compositor's actual work, distinct from idle waiting.
- **Wait-for-damage duration:** How long the compositor waited before starting.
- **Frame scheduling intent:** Damage-driven vs time-driven capture policy.
- **Whether damage was genuine:** Clock tick vs window resize produce the same protocol flow.

4.2 What the Compositor Knows But Does Not Expose

1. When it started the copy operation
2. When the copy completed (GPU fence or memcpy return)
3. Whether damage was present when the capture request arrived
4. The output's current refresh rate

5. Addressing Concerns from Issue #309

5.1 "Do you have evidence?" (andri)

Sections 3.1–3.3 provide that evidence: 1,800× variance in request→ready, 6× cross-compositor divergence, and the idle-to-interactive transition on the same compositor.

5.2 "The compositor doesn't know the copy duration" (mahkoh)

Resolved. The protocol specification states: *"Called as soon as the frame is **copied**, indicating it is available for reading."* Implicit sync (DMA-BUF) means kernel-enforced GPU fence ordering — the compositor waits (automatically) for the GPU to finish. The timing information exists.

5.3 Relevance of Prior Proposals

Both reviewers have proposed timing feedback protocols:

- **Issue #208** (andri): Frame submission deadline event for capture clients
- **Issue #227** (mahkoh): Client→compositor processing duration

The principle of cross-boundary timing visibility is accepted by both parties.

6. Proposal: Composable Capture Feedback

Each compositor has different architecture. A single mandated metric will not fit all. Instead, **optional composable events** that compositors provide based on what they can measure:

Event 1: `capture_begin` (timing)

Timestamp when the compositor began the copy operation. Client computes `copy_duration = ready_time - capture_begin`. Optional — compositors that cannot determine this omit it.

Event 2: `no_damage` (state)

Sent when the frame has no new content since the previous frame. Client can skip encoding. Trivially implementable — every damage-driven compositor already tracks this internally.

Event 3: `source_refresh` (metadata)

Current refresh rate in millihertz. Addresses the core concern from issue #208 (frame pacing). Optional — VRR compositors may omit it.

> **Design Principle**
>
> Clients handle missing events gracefully — falling back to current behavior. No existing compositor or client needs modification. New capabilities appear incrementally as compositors add support.

7. Appendix: Compositor Support Matrix

Compositor	ext-image-copy-capture	Capture path
Sway 1.10+ (wlroots 0.18+)	Yes	Direct Wayland protocol
Hyprland 0.40+	Yes	Direct Wayland protocol
Niri	Yes	Independent implementation
labwc 0.9.4+	Yes	wlroots-based
KDE/KWin 6.x	**No**	PipeWire via zkde_screencast
GNOME/Mutter	**No**	PipeWire via ScreenCast portal

Lamco Development · lamco.ai · March 2026

Data collected with lamco-probe. Raw JSON data available in the repository under `data/309/`.

This report was prepared to provide empirical evidence for wayland-protocols issue #309. The data, analysis, and tooling are available for review and reproduction.