Energy Efficient WiFi Display*

Xinyu Zhang

http://xyzhang.ece.wisc.edu

Assistant Professor

Department of Electrical and Computer Engineering
University of Wisconsin-Madison

*Joint work with Chi Zhang (UW-Madison) and Ranveer Chandra (Microsoft Research)
WiFi Display (Miracast)

- A screen cast technology
  - Specified by the WiFi Alliance
  - Source mirrors its screen to sink via a WiFi Direct connection

**Local streaming mode**

**Tethered streaming mode**

![Diagram showing the difference between local and tethered streaming modes.]

- **Application** ➔ **Screen composer** ➔ **H.264 codec** ➔ **WiFi-Direct interface**
WiFi Display (Miracast)

- Miracast is gaining popularity
  - Source: supported by Android 4.2+, Windows Phone 8.1
  - Sink: many HDMI adapters available
  - Supporting applications without modification
Miracast ≠ Chromecast

- Chromecast
  - Google’s proprietary technology
  - Source redirects video links to sink; Sink (Chromecast dongle) directly downloads video from Internet
  - Need application support
Energy Efficiency of Miracast

- Why it matters?
  - The Miracast source is usually a battery-powered mobile device
  - Power consumption on Galaxy Nexus: up to 2.3 W

- Didn’t the industry (WiFi Alliance) do anything about it?
  - Yes! Video frame skipping (VFS)
    Stop video transmission opportunistically when screen is static.

- How effective is it?
Miracast Source Power Consumption

- Measured on a Galaxy Nexus smartphone, running Android 4.2.2
- Power consumption dominated by **network and codec operations**
Modeling Miracast Power Consumption

Network power consumption:

\[ P_{\text{network}}(Ch, B_i) = P_{\text{static}}(Ch, B_i) + P_{\text{dynamic}}(Ch, B_i) \]

- Power to keep NIC ready
  - Obtained via factory calibration
- Additional power needed for Tx/Rx
  - Depends on factory calibrated factors and two run-time factors (contention intensity and data-rate)

(i) Channel & interface

(ii) Contention intensity

(iii) Video data rate
Modeling Miracast Power Consumption

Codec power consumption:

\[ P_{\text{codec}}(R_i, B_i) = \sum_{i=e,d} \left( E_{fi}(W \times H) \cdot R_i + E_{bi} \cdot B_i \right) \]

- **Frame rate**
- **Bit rate**
- **Resolution**
- **Encoding/decoding energy per-bit**
- **Overhead (CPU/mem) energy per-bit**

(i) **Video data rate**

(ii) **Video resolution**

(ii) **Video frame rate**
Existing Solutions for Mobile Energy Efficiency

- **Energy aware mobile video streaming**
  - Proxy-based protocols
  - Server-based protocols

  \{ Reshape video traffic into bursts to create sleeping opportunities for mobile receiver \}

- **Network interface power optimization**
  - Scheduling protocols to maximize sleep opportunities
Energy Efficient Miracast: Solution Set

- Reducing network/codec operations
  - Adaptive video tail cutting
  - Video pass-through
  - Background suppression

- Improving transmission efficiency
  - Batching and prefetching
  - Off-channel Miracast
  - PSM lock
Video Tails in Miracast

- **Video tail**
  - One “screenshot” cannot be encoded into a single video frame
  - Need to add “tail” frames to gracefully refine video quality
  - In Android Miracast, the tail lasts 1 second (30 frames)

  ![Diagram of video tail](image)

  **1 second**

- This matters only for screens that stay for a while -- Continuous screen update (e.g., video playback) doesn’t have the problem
Video Tails in Miracast

- Video tails improve video quality, but cost energy

Linear growth in power consumption

Marginal improvement in video quality
Adaptive Tail Cutting

- **Principle:** cutting redundant codec operations
  - Instead of always generating 30 tail frames, stop encoding immediately when video quality stops improving

- **When to stop?**
  - Straightforward way: compute PSNR of encoded frames. Limitation: Substantial processing overhead
  - Observation: variance of frame size drops when PSNR plateaus
  - Adaptively cut the tail when variance of frame size is low

![Graph showing PSNR and variance](image)
Adaptive Tail Cutting

- Implementation on Android
Video Pass-through

- Principle: bypass codec operations and offload to sink
  - For streaming apps, the source needs to \textbf{decode} video for screen display, and then \textbf{reencode} it for Miracasting.

- We can bypass such decoding/reencoding operations.

Replace with an H.264 format checker.
Video Pass-through

- Hiding video pass-through from the sink
  - Timestamp conversion
    * Convert video’s native timestamp to Miracast timestamp
  - Sequence number conversion
    * Native sequence number is not readable (encoded in unknown format)
    * Leverage H.264 configuration frames to reset sequence number, when video cast starts/ends
**Batching and Prefetching**

- **Principle**: amortizing transmission overhead
  - Applicable to locally stored video
  - Send entire batch *before* the due time of first frame
  - A *tradeoff* between saving energy and wasted transmission

[Graph showing estimated power consumption versus batch size with different interruption frequencies indicated.]
Off-channel Miracast

- **Principle**
  - Select an energy-efficient channel for the Miracast link, independent of any infrastructure (access points)

- **Approach**
  - Energy-aware off-channel selection
    * Model-driven approach to determine energy-optimal channel
    * Ping delay $\rightarrow$ Contention intensity $\rightarrow$ Channel-dependent power consumption in Miracast
Other Optimizations

- Optimizing WiFi Power Saving Mode (PSM) tails
  - WiFi interface remains on for a while, after each transmission
  - PSMlock: shorten PSM tail and let it sleep immediately, leveraging Miracast’s periodic traffic pattern

- Optimizing invisible background traffic
  - Silent background audio
  - Background image layers
Implementation

- Application-transparent implementation
  - Based on Android 4.2.2 Miracast framework
  - Any application can run on top
    * Video streaming: local streaming, tethered streaming
    * Bursty video: Presentation (30s per slide), map navigation (1s per frame)
  - Source is a Galaxy Nexus phone
Effectiveness of PSMlock

- PSMlock saves up to 5.2%
Effectiveness of Video Tail Cutting

- Power saving depends on traffic intensity and burstiness
- Tail-cutting does not affect video quality

![Graph showing power saving and PSNR vs. fraction of frames for different tail cutting methods.](image)
Effectiveness of Video Pass-through

- Pass-through saves substantial processing power

- Pass-through plus prefetching reduces latency
Effectiveness of Off-channel Miracast

- Model-driven approach can effectively select top-3 most energy efficient channels

- Power saving (in tethered streaming mode) depends on contention intensity
System-level Power Saving

- 29% to 61% saving, depending how application refreshes screen

A: no contention
B: severe contention
Conclusion

- Profiling power consumption of WiFi Display (Miracast)
  - Major cost: codec and network
  - A measurement profiling and modeling framework

- Optimizing Miracast energy efficiency
  - System-level mechanisms to improve energy efficiency
  - Optimizing codec+network operations

- Future directions
  - Higher layer: application-aware power optimization
  - Lower layer: integrating with WiFi Direct’s sleep scheduling
Thank you!