Making JavaScript Better By Making It Even Slower

Maciej Swiech, Peter Dinda
Northwestern University
Empathic Systems Project
www.empathicsystems.net

What will we talk about?

- We were able to reduce energy spent on mobile browsing, extending battery life
- In most cases, we are able to accomplish this with little to no effect on the user
- We suggest ways to implement this effect

Outline

- Motivation / Background
- Key Idea – throttling
- Enabling technology (TameJS)
- JSSlow Proxy
- Offline Studies
- User Study
- Conclusions

Outline

- Motivation / Background
- Key Idea – throttling
- Enabling technology (TameJS)
- JSSlow Proxy
- Offline Studies
- User Study
- Conclusions

Modern Browsing

- Modern web sites rely on enabling technologies like JavaScript
- Implementation of a Model-View-Controller
- Much correctness/efficiency research
  - Google Closure Compiler
  - S5 Semantics [Politz et al. DLS ‘12]

JavaScript

- Integral to modern website design
  - Dynamic and interactive user environment
- Event-based
  - Registered handlers – `onClick()`, `onLoad()`, etc
  - Interpreter waits for event to occur
- Runtime
  - Single-threaded

JavaScript: mobile

- Buggy code detrimental to user experience
- Power, energy, and battery lifetime considerations
  - Transmission and interpretation significant portion of energy spent on mobile browsing
  - Amazon – 16%
  - YouTube – 20%
  - [Thiagarajan, N. et al. WWW ’12]

JavaScript: mobile

- How can we reduce energy?
  - Code minification / obscuring
  - Compression schemes

- Reduce transmission energy, but not interpretation and running energy

Outline

- Motivation / Background
- Key Idea – throttling
- Enabling technology (TameJS)
- JSSlow Proxy
- Offline Studies
- User Study
- Conclusions

Throttling

- We argue JavaScript is running faster than it needs to be
- What if we throttle interpretation?

Throttling: methods

- DVFS
- Thread scheduling
- Inserting sleep()

Throttling

- Idea – insert ‘sleep()’ calls at key control-flow points in code
  - `if`, `for`, `while`, function definitions
  - Easily identifiable
  - Likely to be repeated

- Reduce energy while maintaining user satisfaction

Throttling

- "Race to the finish" computation?
- *Dwell Time* = time spent on a site

$$Energy = Dwell\ Time * \ Power$$

- Doesn't capture event-based model

- Speed of execution ≠ *dwell time*
  - Power savings → Energy savings

Outline

- Motivation / Background
- Key Idea – throttling
- Enabling technology (TameJS)
- JSSlow Proxy
- Offline Studies
- User Study
- Conclusions

Throttling JavaScript

- No native `sleep()`!
  - Single-threaded event-based model

TameJS

- JavaScript extension compiler
- Based on Tame C++ framework [Krohn et al. USENIX ATC ‘07]
- Extends JavaScript with 2 primitives
  - `await`
  - `defer`

- Designed to make event programming easier to develop in JavaScript

TameJS: example

```
for (var i = 0; i < 5; i++) {
     console.log("hello");
}
```

TameJS: example

```
for (var i = 0; i < 5; i++) {
     console.log("hello");
}
```

hello hello hello hello hello

TameJS: example

```
for (var i = 0; i < 5; i++) {
     setTimeout(console.log("hello"),1000);
}
```

TameJS: example

```
for (var i = 0; i < 5; i++) {
    setTimeout(console.log("hello"),1000);
}
```

wait 1 second...

TameJS: example

```
for (var i = 0; i < 5; i++) {
     setTimeout(console.log("hello"),1000);
}
```

wait 1 second…

hello hello hello hello hello

TameJS: example

```
for (var i = 0; i < 5; i++) {
      await{setTimeout(defer(), 1000);}
      console.log("hello");
}
```

TameJS: example

```
for (var i = 0; i < 5; i++) {
      await{setTimeout(defer(), 1000);}
      console.log("hello");
}
```

wait 1s, hello, *wait 1s,* hello, …

TameJS → Throttling

```
await{ setTimeout(defer(), time);}
```

- This “sleep()” causes interpreter to pause → yield
- OS can deschedule interpreter → HLT
- If CPU idle → C-STATE can be lowered

TameJS → Throttling

- How long to sleep?

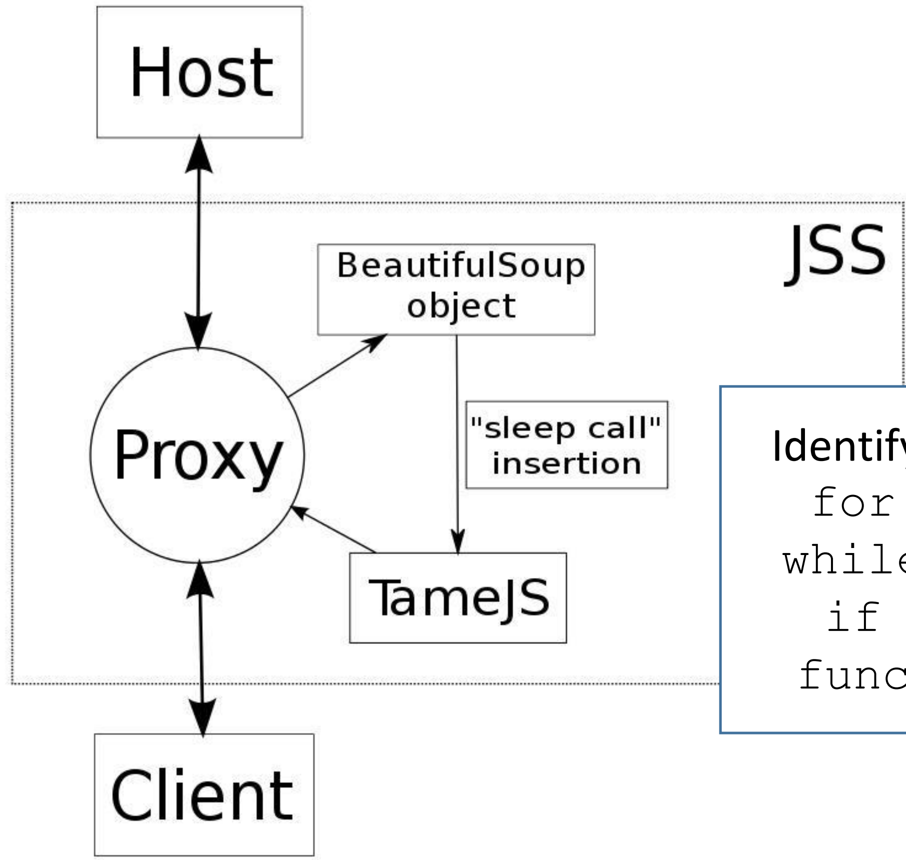
- Tested delays of 1,2,5,10,25,100ms
  - Once any sleep injected, reduction of CPU util

- Chose 1ms to cause least impact on user satisfaction

Outline

- Motivation / Background
- Key Idea – throttling
- Enabling technology (TameJS)
- JSSlow Proxy
- Offline Studies
- User Study
- Conclusions

JSSlow



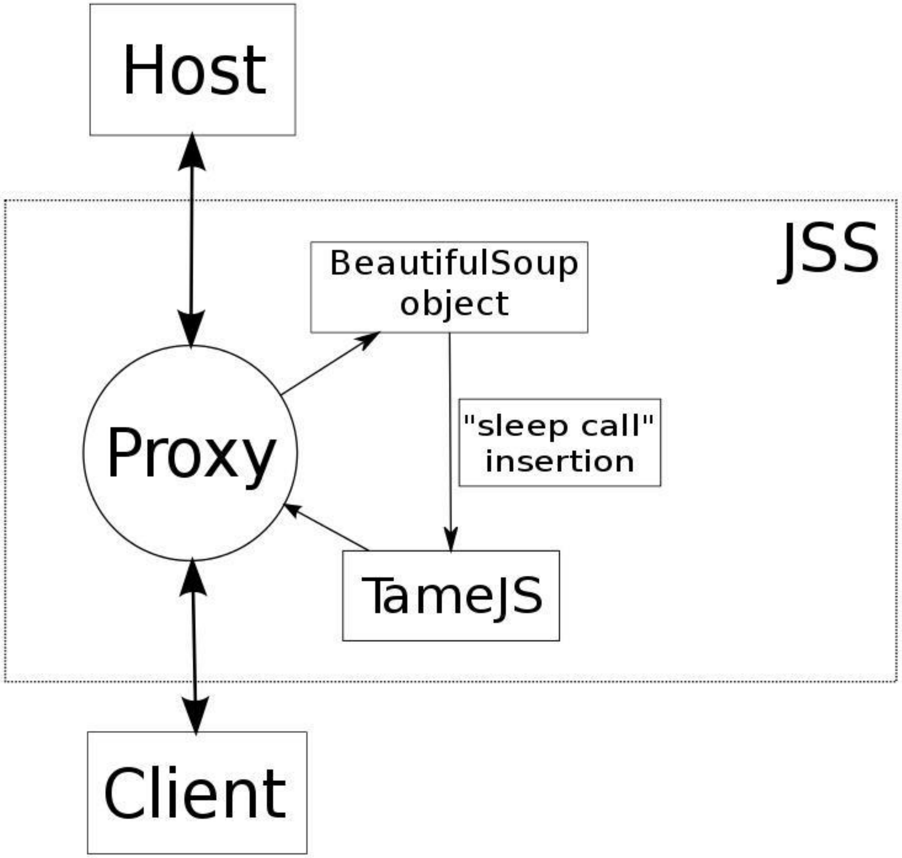
Identify:

`for`
`while`
`if`
`func`

JSSlow: architecture

- Proof-of-concept HTTP proxy
  - Evaluate throttling claims
  - Insert between user and web site
- Based on TinyHTTP proxy
  - Python
  - Used in previous studies to provide satisfaction overlay [J. Miller et al. INFOCOM '10]
- BeautifulSoup library
  - HTML AST creation
  - Fast identification of `<script>` nodes

JSSlow: architecture

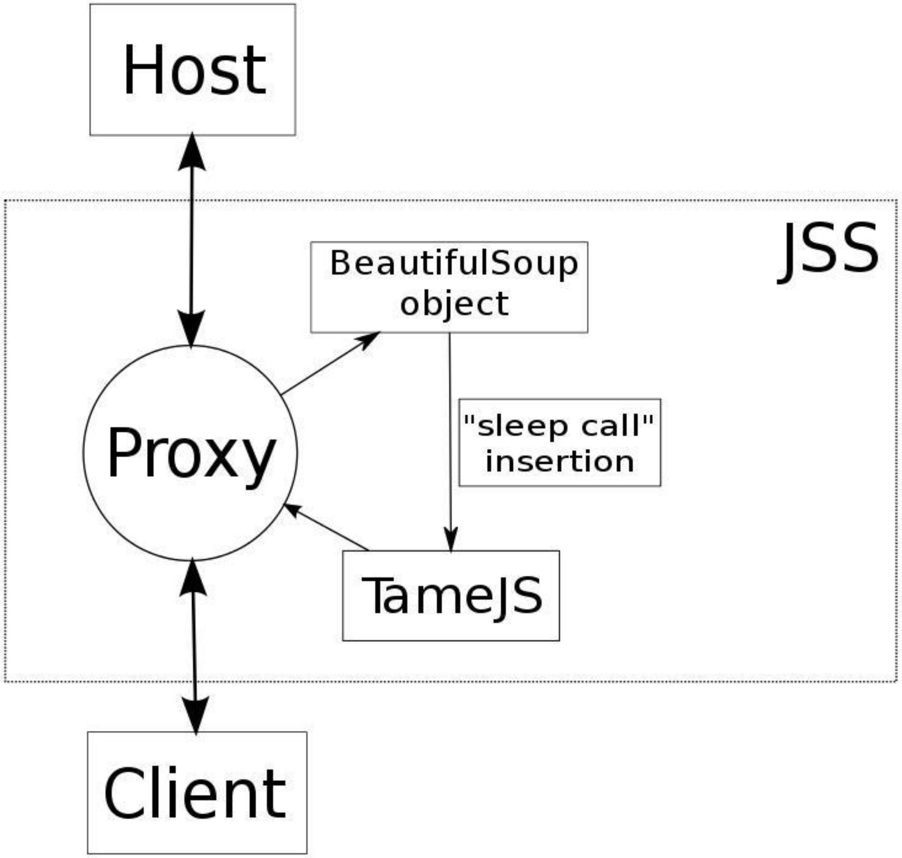



```
<script>
  ...
  var i = getThing();

  for (j = 0; j < 3; j++) {
    do_a_thing();
  }

  while (j == 4) {
    do_another_thing();
  }
  ...
</script>
```

JSSlow: architecture




```
<script>
  ...
  var i = getThing();

  for (j = 0; j < 3; j++) {
    await{setTimeout(defer(),1000);}
    do_a_thing();
  }

  while (j == 4) {
    await{setTimeout(defer(),1000);}
    do_another_thing();
  }
  ...
</script>
```

Outline

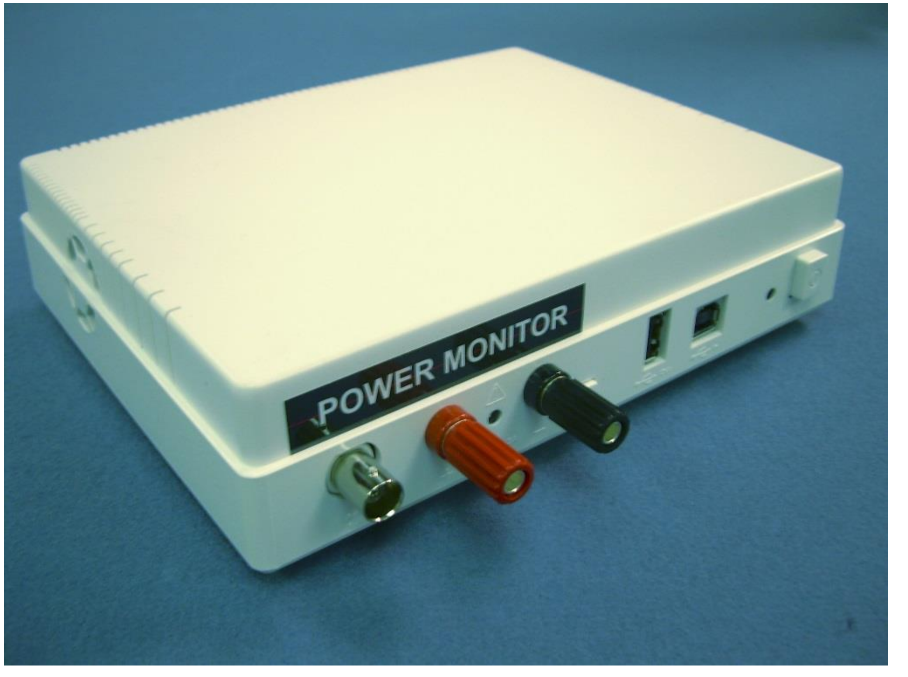
- Motivation / Background
- Key Idea – throttling
- Enabling technology (TameJS)
- JSSlow Proxy
- Offline Studies
- User Study
- Conclusions

Evaluation: offline studies

- Top-k study
  - Studied effect on most popular web sites
  - Automated page-loading
- Advertising / Buggy JavaScript study
  - Studied effect on advertising JavaScript
  - Measured upper bound using crafted bugs

Offline: testbed

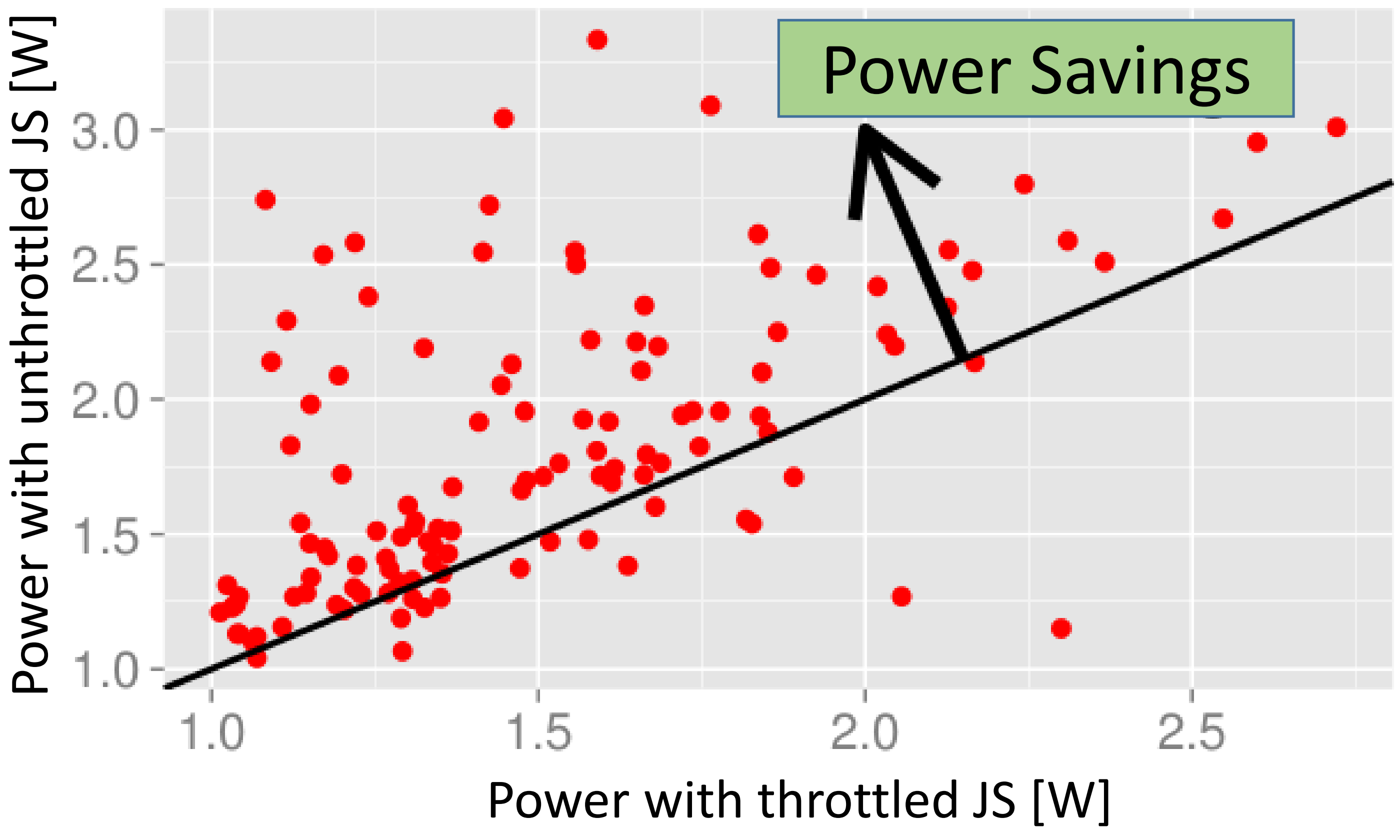
- Galaxy Nexus phone
- Android 4.0.4
- Monsoon power monitor
  - Bypass battery



Offline: top-k study

- 120 most popular sites gathered from Google Ad Planner
- Each site allowed to run for a *dwell time* of 60 seconds
  - Allow site to load and settle
- Runs repeated with throttling enabled and disabled in proxy

Offline: 5% power reduction for top-k



Offline: advertising and bugs

- 50 JavaScript ads manually extracted from random sample of top 120 sites
- Each ad run for 60 seconds
- Runs repeated with throttling enabled and disabled in proxy

- Crafted infinite loop to estimate upper bound

Offline: ad and bug results

- 52% reduction in power during infinite loop
  - Page usability restored

- Average 10% reduction in power for advertisements

Outline

- Motivation / Background
- Key Idea – throttling
- Enabling technology (TameJS)
- JSSlow Proxy
- Offline Studies
- User Study
- Conclusions

Evaluation: user study

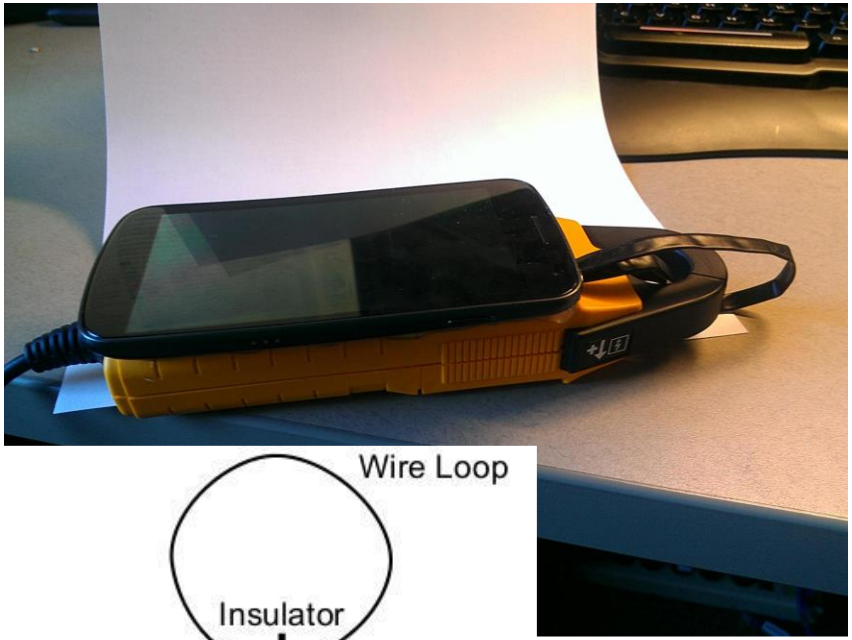
- Designed a double blind user study to evaluate effects of both real-time energy effects and user satisfaction
- Chose first 20 users who responded to call for study

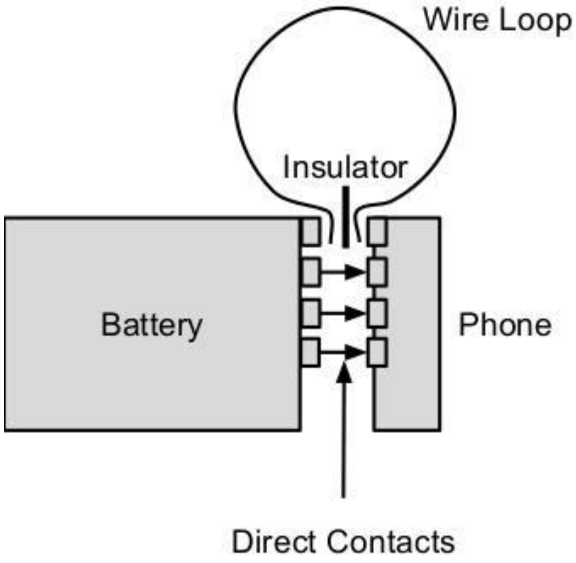
User Study: design

- User establishes a baseline on non-throttled phone, familiarizing themselves with browser
- User would complete each task
  - ‘low interactivity’ – read / comment on CNN, read / comment on FaceBook
  - ‘high interactivity’ – play JavaScript game of Snake
- Every 30 seconds, user prompted to rate satisfaction
- Proxy randomly chose whether to throttle

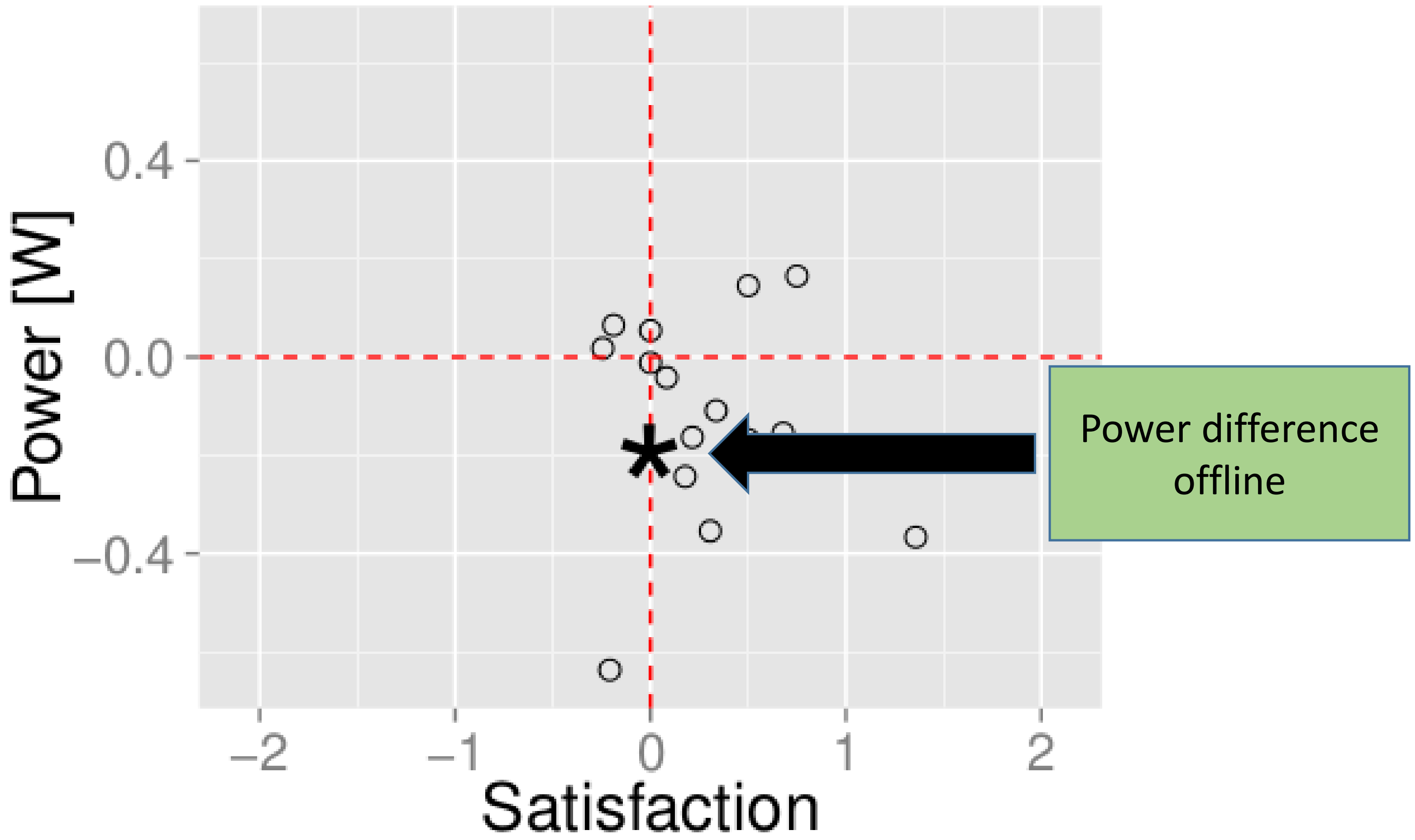
User Study: testbed

- Galaxy Nexus phone
- Android 4.0.4
- Fluke i30 current clamp
- RadioShack 22-812 DMM
  \+ QtDMM

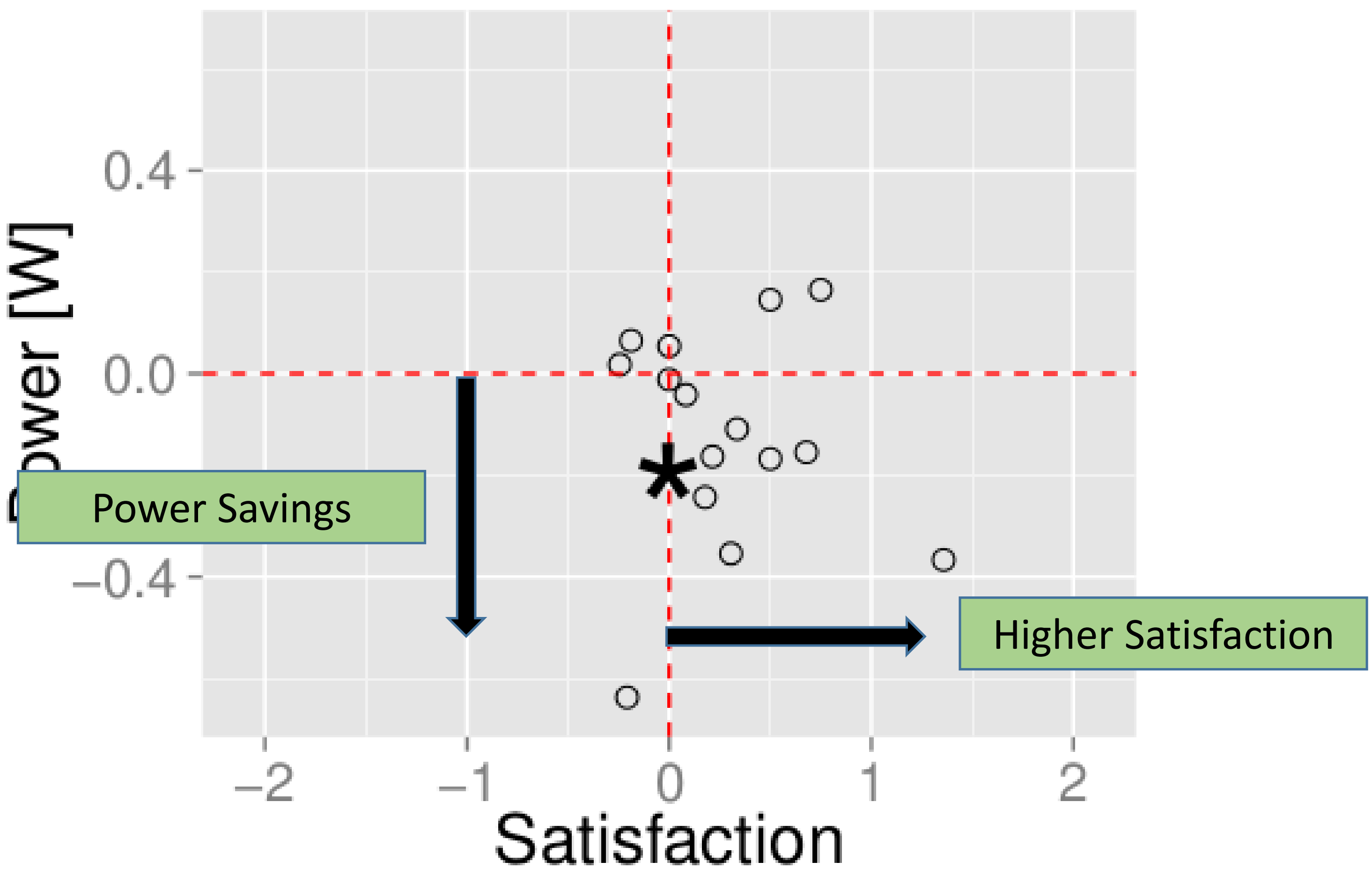




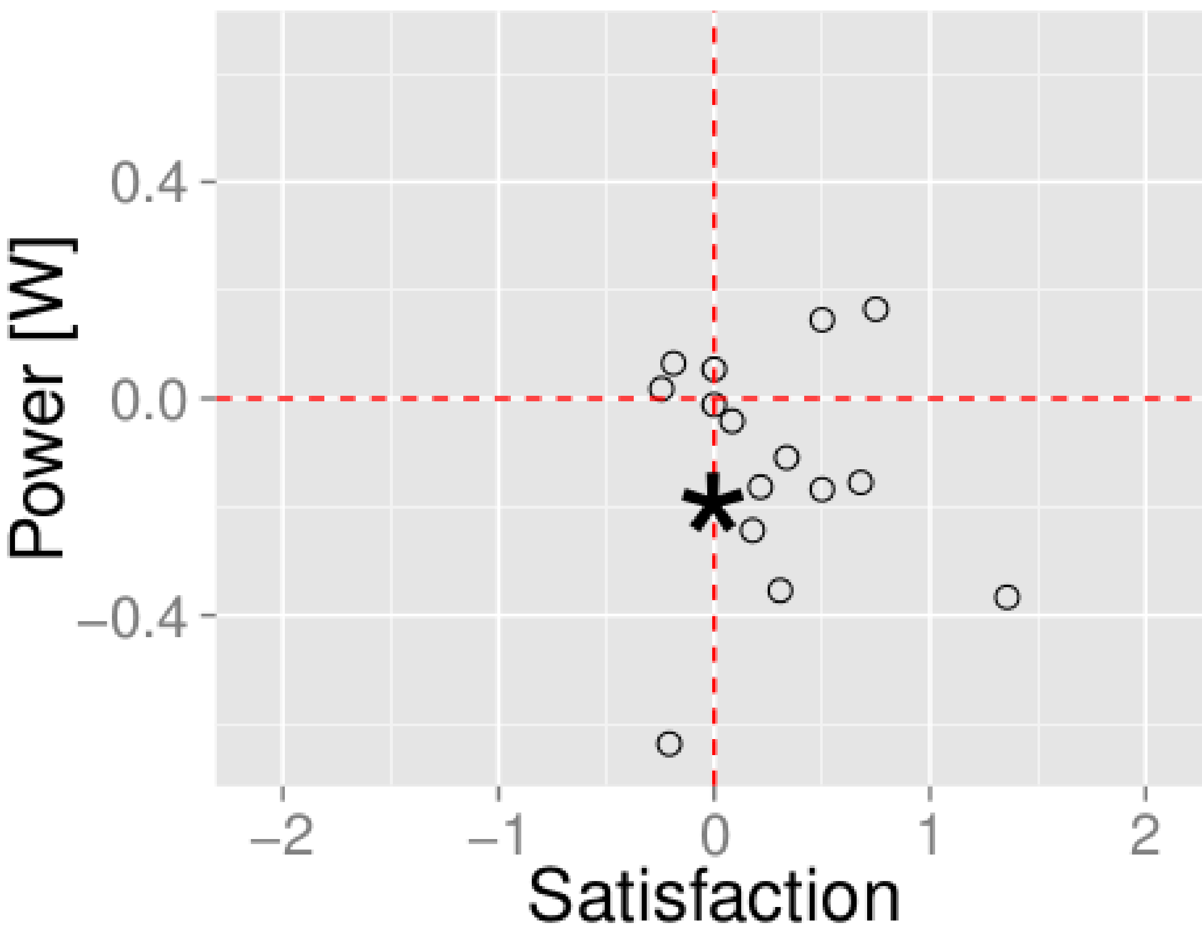
User Study: results



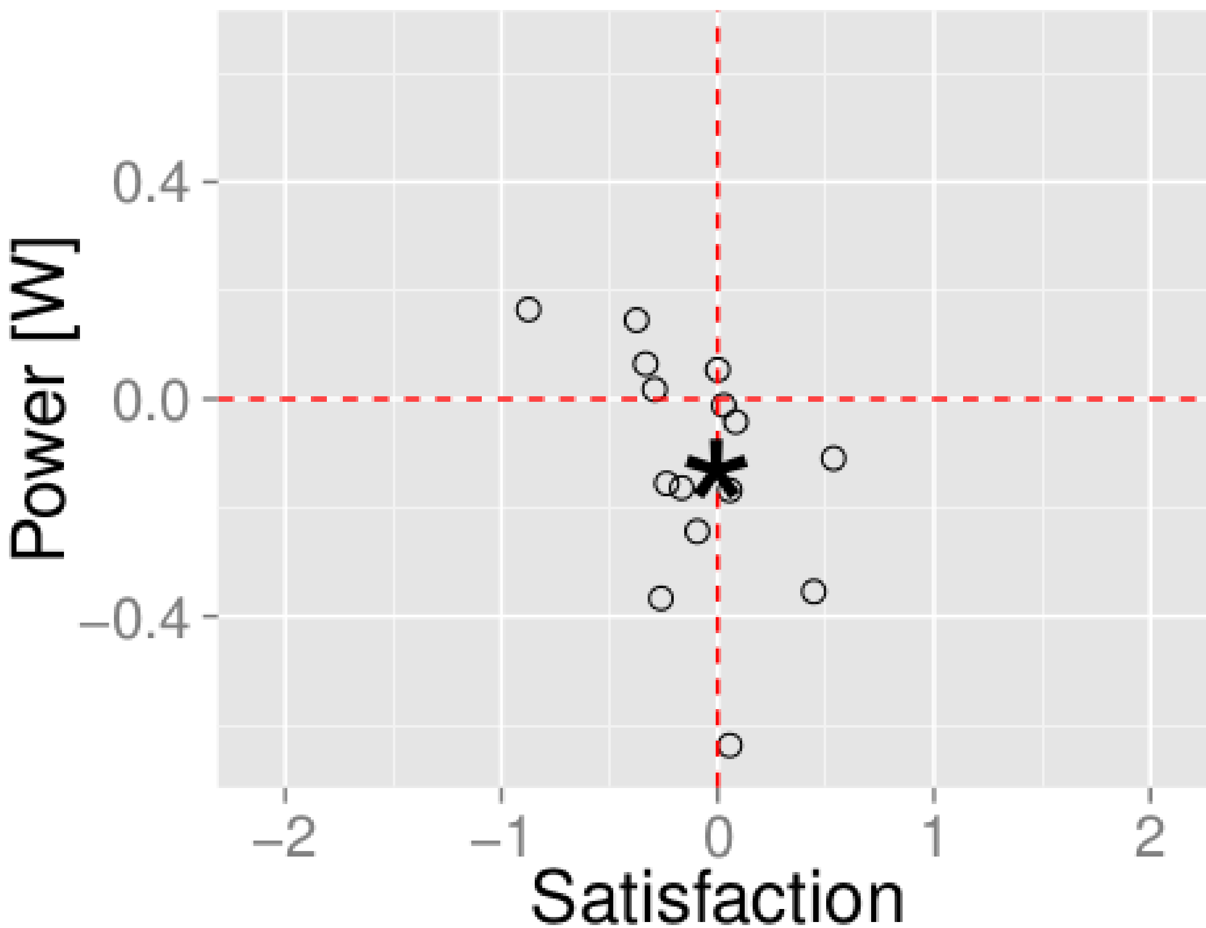
User Study: results



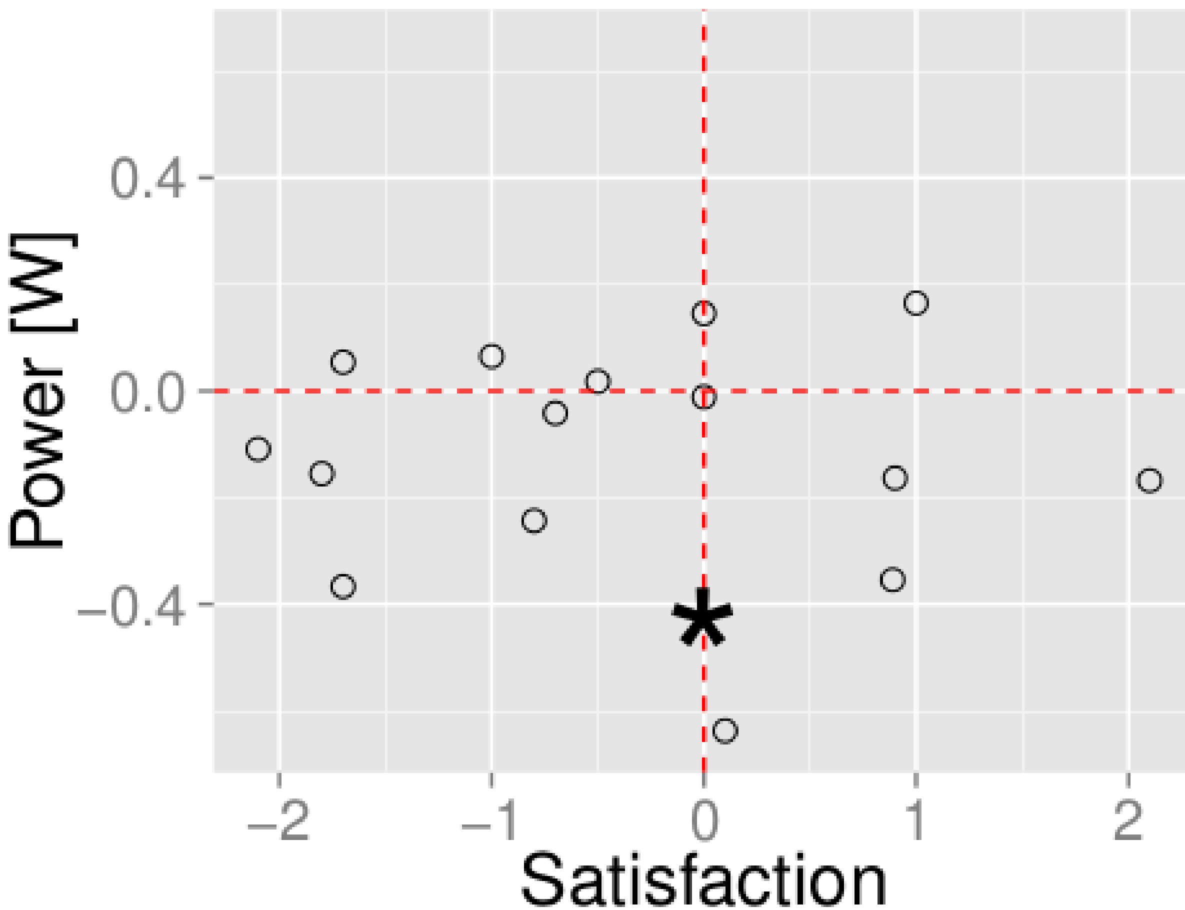
User Study: CNN – lower power



User Study: FaceBook – lower power



User Study: Snake[1] – varied



[1]http://snake.alexthorpe.com

User Study: results

- Low interactivity
  - Small change in satisfaction for CNN
  - Mixed change in satisfaction for FaceBook
  - Average power reduction: 3.8%

- High interactivity
  - No power savings
  - Very varied satisfaction

Evaluation: proxy limitations

- Increased download size
  - TameJS transformation + runtime library
- Decreased performance
  - TameJS transformation can lead to 1-2% performance loss
- Coarse-grained control
- Missed opportunities
  - Non-locally sourced scripts (advertising)
  - TameJS compilation errors

Outline

- Motivation / Background
- Key Idea – throttling
- Enabling technology (TameJS)
- JSSlow Proxy
- Offline Studies
- User Study
- Conclusions

Results

- By throttling JavaScript we are able to reduce energy during mobile browsing by 3-10%
  - Underestimation due to implementation

- This reduction comes at little to no cost to the end-user for low-interactivity sites

- More controls needed for high-interactivity sites

Future Work: throttling

- Most of JSSlow’s limitations can be mitigated by implementing throttling in the JavaScript engine
  - Default throttle settings
  - Crowdsourced database
  - JavaScript APIs

- SpiderMonkey and V8
  - Rudimentary implementation

- Throttling reduces energy by 3-10%
- Throttling comes at little to no cost for the user
- Proxy proof-of-concept, Engine augmentation ideas

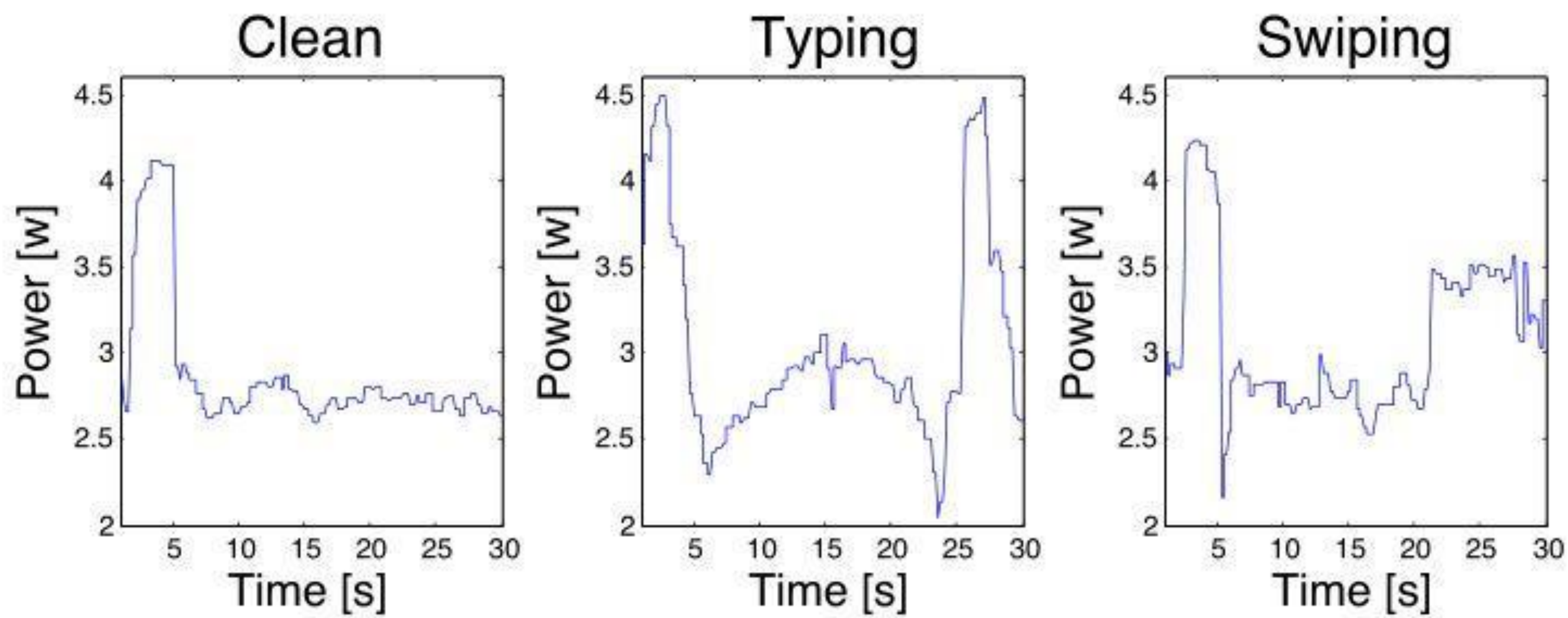
Maciej Swiech <mswiech@u.northwestern.edu>

http://eecs.northwestern.edu/~msw978

Prescience Lab: www.presciencelab.org

Empathic Systems Project: www.empathicsystems.org

Android interactive governor



JSSlow: algorithm

```
// create AST of the incoming html
   html-copy = BeautifulSoup(incoming-html)
   sleep = "await { setTimeout(defer(), g_slow); }"
   // iterate over all <script..>..</script> fields
   for script in html-copy:
       script-copy = script
       // fetch local scripts
       if script.has_tag("src") && src.is_local():
           script-copy = fetch(src.address)
       insert-at(sleep, ["while","for","if","function"])
       try:
           script-copy = tame-compile(script-copy)
       except:
           // if compilation failed, just skip
           continue
       script = script-copy
   return html-copy
```