RadioJockey: Mining Program Execution to Optimize Cellular Radio Usage

Pavan Kumar, Ranjita Bhagwan, Saikat Guha, Vishnu Navda, Ramachandran Ramjee, Dushyant Arora, Venkat Padmanabhan, George Varghese

Microsoft Research India
Problem Context: Overheads in Cellular Radio Usage

State transitions based on:
(1) traffic volume
(2) operator chosen timers

Power Consumption

Signaling

Transition # control messages

IDLE \(\rightarrow\) DCH 30
DCH \(\rightarrow\) IDLE 2

Latency

Transition Secs
IDLE \(\rightarrow\) DCH 2
DCH \(\rightarrow\) IDLE 20
Existing Radio-tail Optimizations

1. Amortize tail overhead by shaping traffic
   a) TailEnder [IMC 09]

2. Adapt tail using Fast-dormancy
   a) Based on application hints – TOP [ICNP 10]
   b) Based on client-side idle timers – Falaki et al. [IMC 10]
Existing Radio-tail Optimizations

1. Amortize tail overhead by shaping traffic
   a) TailEnder [IMC 09]  
      Requires app changes

2. Adapt tail using Fast-dormancy
   a) Based on application hints –
      Requires app changes + developer awareness
   b) Based on client-side idle timers -
      Commonly used in many smartphones (3-5 sec timers)
Fast Dormancy Woes

Disproportionate increase in signaling traffic caused due to increase in use of fast-dormancy

“Apple upset several operators last year when it implemented firmware 3.0 on the iPhone with a fast dormancy feature that prematurely requested a network release only to follow on with a request to connect back to the network or by a request to re-establish a connection with the network …”

What's really causing the capacity crunch? - FierceWireless
Problem #1: Chatty Background Apps

- No distinctive knee
- High mispredictions for fixed inactivity timer
Problem #2: Varying Network Conditions

- Signal quality variations and handoffs cause sudden latency spikes
- Aggressive timers frequently misfire

CDF of inter-packet times for Lync application for different network conditions
Objectives

• Design a fast-dormancy policy for long-standing background apps which
  – Achieves energy savings
  – Without increasing signaling overhead
  – Without requiring app modifications
When to Invoke Fast Dormancy?

Energy savings when $t_s \geq 3$ sec and fast dormancy is invoked immediately after end of session.
Problem: predict end of session (or onset of network inactivity)

Idea: exploit unique application characteristics (if any) at end of sessions

Typical operations performed:

- UI element update
- Memory allocation or cleanup
- Processing received data

System calls invoked by an app can provide insights into the operations being performed
Predicting onset of network inactivity

- **Technique:** Supervised learning using C5.0 decision trees
- **Data item:** system calls observed immediately after a packet (encoded as bit-vector)
- **Label:** ACTIVE or EOS
Decision tree example

Application: gnotify

Rules:
(DispatchMessage & !send) => EOS
!DispatchMessage    => ACTIVE
(DispatchMessage & send) => ACTIVE
RadioJockey System

Offline learning

System Calls + Network Traffic → Training using C5.0

Runtime Engine

App System Calls + Packet timestamps → Tree-matching (run-time) → Fast Dormancy

App 1 Rules

App k Rules

Cellular Radio Interface
Evaluation Overview

1. **Trace driven simulations** on traces from 14 applications (Windows and Android platform) on 3G network
   - Feature set evaluation for training
   - variable workloads and network characteristics
   - 20-40% energy savings and 1-4% increase in signaling over 3 sec idle timer

2. **Runtime evaluation** on 3 concurrent background applications on Windows
Energy drain and signaling overhead

Energy consumed normalized to a 3-second idle timer approach

Signaling overhead normalized to a 3-second idle timer approach
Runtime Evaluation with Concurrent Background Applications

- 22-24% energy savings at a cost of 4-7% signaling overhead
- Marginal increase in signaling due to variance in packet timestamps

<table>
<thead>
<tr>
<th>Applications</th>
<th>Energy Savings (%)</th>
<th>Signaling Overhead (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outlook</td>
<td>24.03</td>
<td>4.47</td>
</tr>
<tr>
<td>GTalk</td>
<td>24.07</td>
<td>4.57</td>
</tr>
<tr>
<td>Lync</td>
<td>24.14</td>
<td>0</td>
</tr>
<tr>
<td>All</td>
<td>22.8</td>
<td>6.96</td>
</tr>
</tbody>
</table>
Summary

• RadioJockey predicts onset of network inactivity using system calls invoked by background apps

• Requires no modifications to existing apps – legacy, native and managed apps

• Achieves energy savings of 20-40% with marginal increase in signaling overhead
Backup Slides
Predict using only network features

• Features: IP, ports, TCP flags, HTTP headers

• Performance:
  – Energy savings only for simple apps
  – No good rules for complex apps (Outlook and Lync)
  – Cannot handle apps that use encryption
Varying networks and workloads

Energy consumed normalized to a 3-second idle timer approach
Feature Space Exploration and Choice of Window Size

- PrevState feature captures temporal state information
- Adding PrevState into learning boosted savings
- $t_w$ of 0.5 seconds sufficient for most applications
Understanding Fast Dormancy Feature

• Client controlled

• Tail energy reduced to ~1.5J

• Without network support
  – RRC connection torn down
  – DCH/FACH to IDLE
  – Ramp-up costs up to 30 msgs

• With network support
  – Ramp-down to PCH instead of IDLE
  – Ramp-up to DCH incurs 12 msgs