Mobile Cloud Offloading on Crowdsensing Platforms

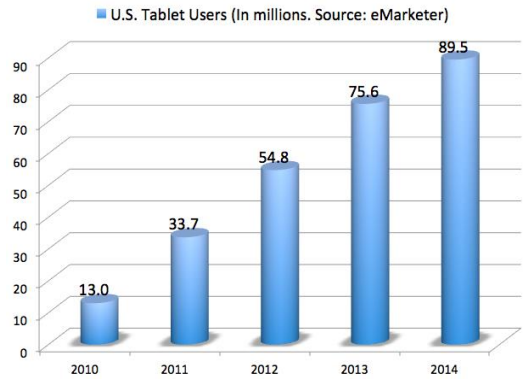
Ting Yi Lin

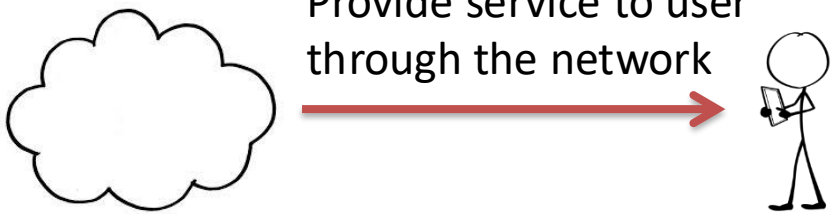
Outline

- Introduction
  - Efficient Crowdsensing System With Offloading
  - Offloading Applications To Cloud
  - Contributions
- Offloading Applications To Cloud
  - Architecture
  - Problem Statement
  - Algorithm
  - Experiments
- Offloading Event Analysis Algorithms: Using IsCrowded And IsNoisy As Case Studies
  - Implementation
  - Case Studies
  - Evaluations
- Conclusion And Future Work

Population of Mobile Device, Wireless Network, and Cloud Platform

- Mobile devices are more and more popular
  - Smart phone, tablet, etc
  - 1.75 billion smartphone users (2014)[1]
- Wireless network coverage
  - The number of access points and the bandwidth are increasing

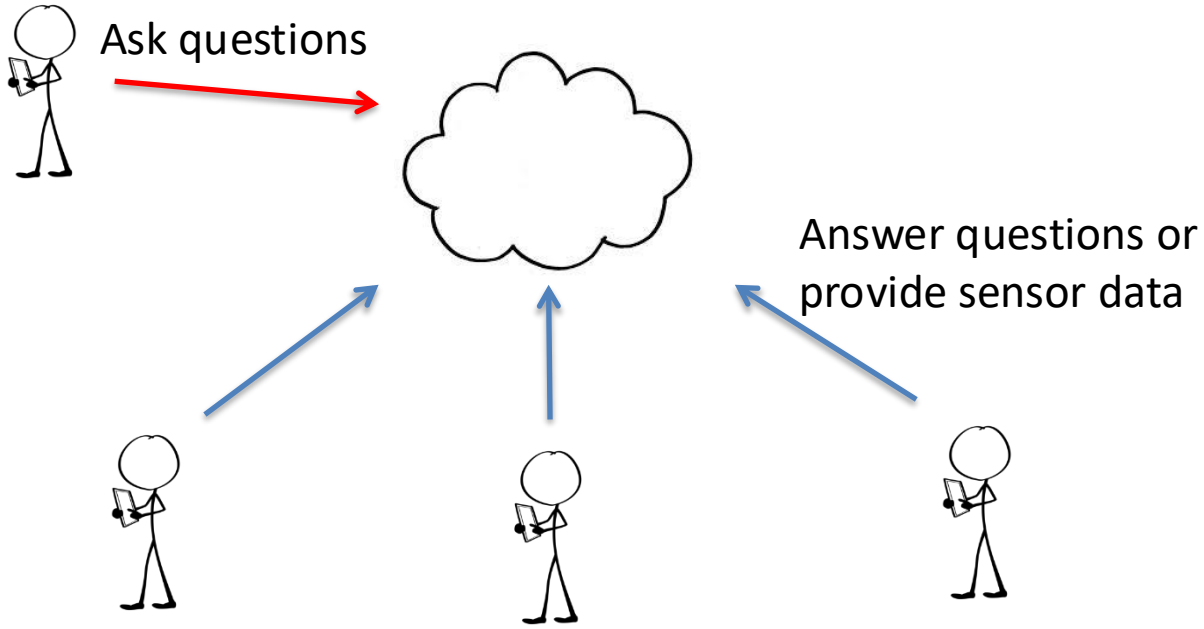




[1] http://www.emarketer.com/Article/Smartphone-Users-Worldwide-Will-Total-175-Billion-2014/1010536

What Is Crowdsensing ?

- Smartphones are now equipped with multiple sensors
  - GPS, accelerometer, microphone, gyroscope, etc.
- The crowds provide useful information for each others
  - Events are detected through processing the sensory data



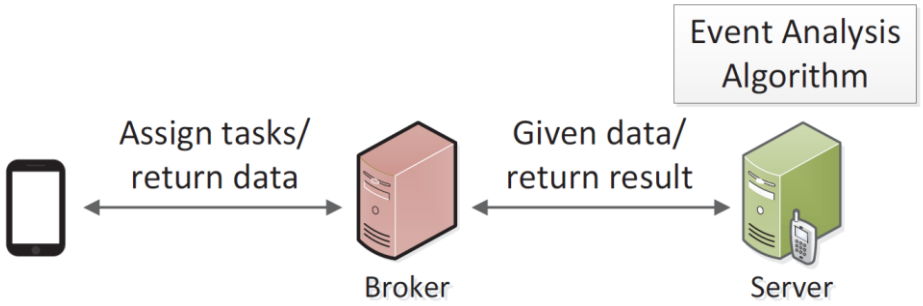
Why Is Crowdsensing Interesting?

- It has many applications

- It has many advantages
  - Users can ask questions which are hard for algorithms to answer
  - Improving the coverage of traditional sensor networks
  - Lowering the deployment and maintenance cost for companies

What Is The Most Efficient Execution Flow?

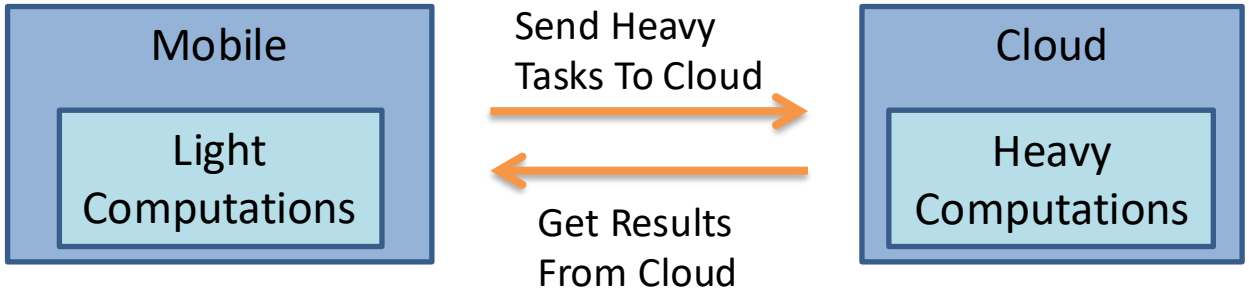
- Most of the crowdsensing systems analyze the sensory data on broker/server
  - It is not efficient
  - The computation capability and network bandwidth of servers and mobiles are different
- Offloading is a good solution



Network Type	Wifi	3G	Local
Current (mA)	9.2619	285.3662	24.0448
Time (ms)	224.99	3559.85	299.34

Mobile Cloud Offloading

- Mobile devices has limited resources
  - CPU, memory, GPU, battery lifetime, etc.
  - Cannot run resource-intensive applications
- Cloud service can be used to address on the issue
  - Cloud servers have large amount of resources
  - Can be accessed easily through network
  - Using powerful server to do the heavy computations



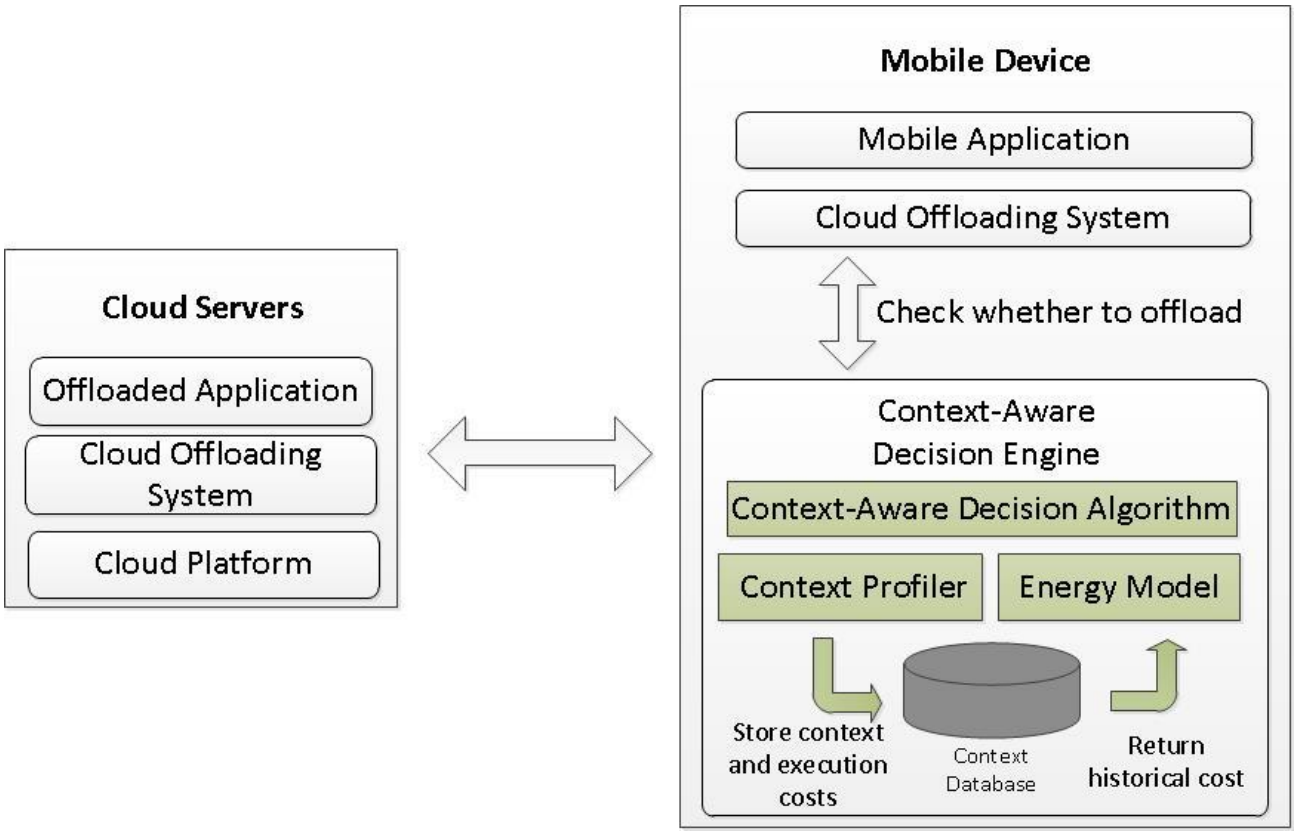
Contributions

- We propose an offloading decision algorithm [MCC'13]
- We develop an APK analysis tool to analyze and modify APK files
- We offload existing third-party applications to show offloading is feasible in existing applications
- We integrate offloading into a crowdsensing prototype system to show the performance gain from offloading [PIMRC'14]

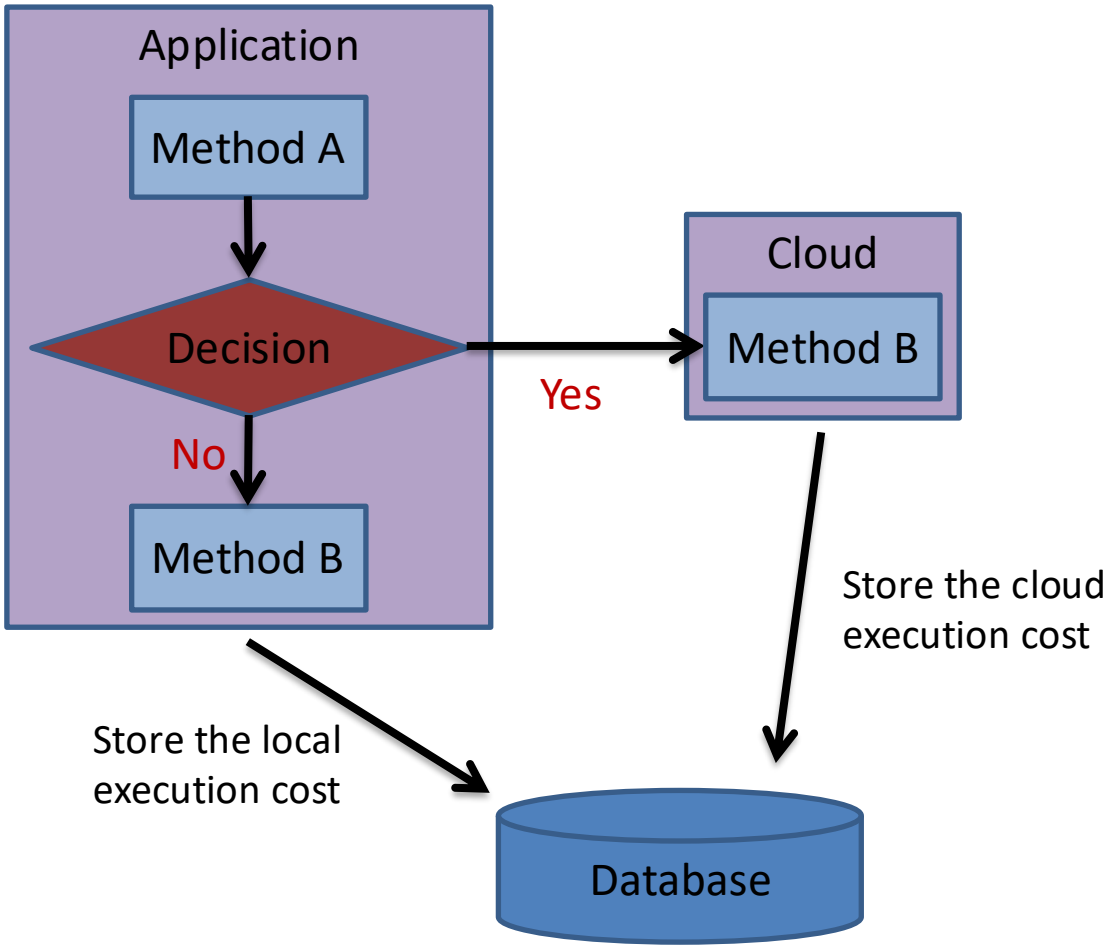
Outline

- Introduction
  - Efficient Crowdsensing System With Offloading
  - Offloading Applications To Cloud
  - Contributions
- Offloading Applications To Cloud
  - Architecture
  - Problem Statement
  - Algorithm
  - Experiments
- Offloading Event Analysis Algorithms: Using IsCrowded And IsNoisy As Case Studies
  - Implementation
  - Case Studies
  - Evaluations
- Conclusion And Future Work

Architecture



System Execution Flow



Why We Need A Decision Engine?

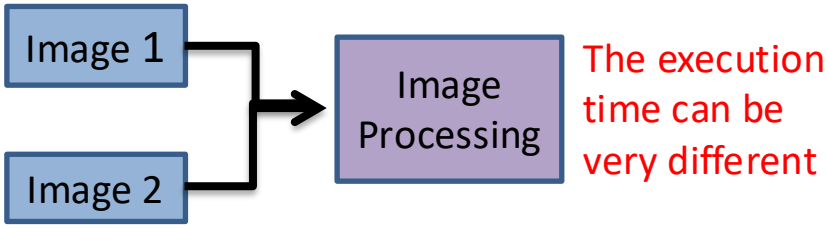
- Offloading cannot always improve performance or reduce energy consumption
  - Transmission energy > computation energy
- Many factors can affect the offloading efficiency
  - Wireless connectivity, mobile CPU usage, etc.
- We aim to minimize the energy consumption on mobile device or improve the application performance
  - We should carefully determine whether to offload

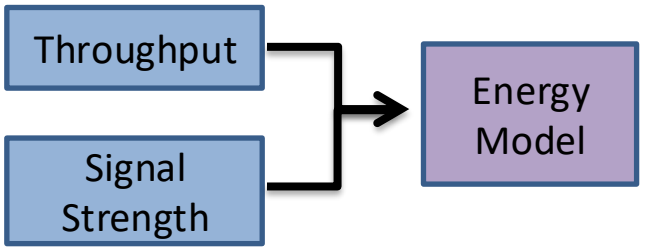
Offload? Not Offload?

- The decision algorithm should not introduce heavy overhead
- We proposed the context-aware decision algorithm to determine whether to offload the computation
  - Context is the information of user and mobile device
    - CPU usage, network connectivity, etc.
- Considered context
  - Signal strength
  - Throughput
  - Time-of-day
  - Location

Why We Use These Four Contexts?

- To predict whether offloading has positive or negative impact
- To estimate the energy consumption of tasks
  - Execution energy and transmission energy
  - Throughput and signal strength can be used to model transmission energy
- Many factors affect the offloading efficiency
  - Data size, transmission time, computation complexity, etc.
  - Time-of-day and location may imply the user behaviors and other factors





Existing Energy Model

- Based on PowerTutor[2]

$$P_{total} = P_{cpu} + P_{comm} + P_{display} + P_{other}$$

$$P_{comm} = P_{WiFi} + P_{Cell}$$

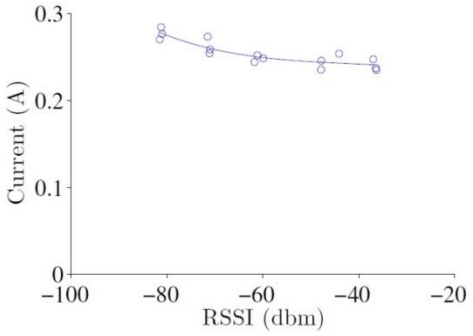
$$P_{WiFi/Cell} = P_{idle} \times \beta_{idle} + P_{trans} \times \beta_{trans}$$

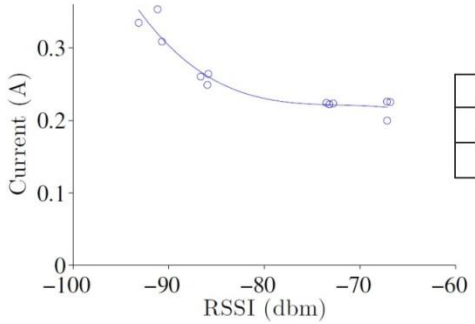
- The model assumes a constant power level and may lead to inaccurate estimations
- We need a better model

[2] L. Zhang et al. , "Accurate online power estimation and automatic battery behavior based power model generation for smartphones." In Proc. of International Conference on Hardware/Software Codesign and System Synthesis, 2010.

Proposed Context-Aware Energy Model

- We measure the mobile device's current flow with Agilent 66321D power meter





MSE	Linear	Quadratic	Cubic
Cellular	5.08×10^{-4}	2.20×10^{-4}	1.77×10^{-4}
WiFi	6.52×10^{-5}	4.35×10^{-5}	4.25×10^{-5}

- Replace communication energy model with our model

$$P_{trans}(S) = \gamma_3 \times S^3 + \gamma_2 \times S^2 + \gamma_1 \times S + \gamma_0$$

$$E_{trans}(S, R, D) = P_{trans}(S) \times \frac{D}{R} \times V$$

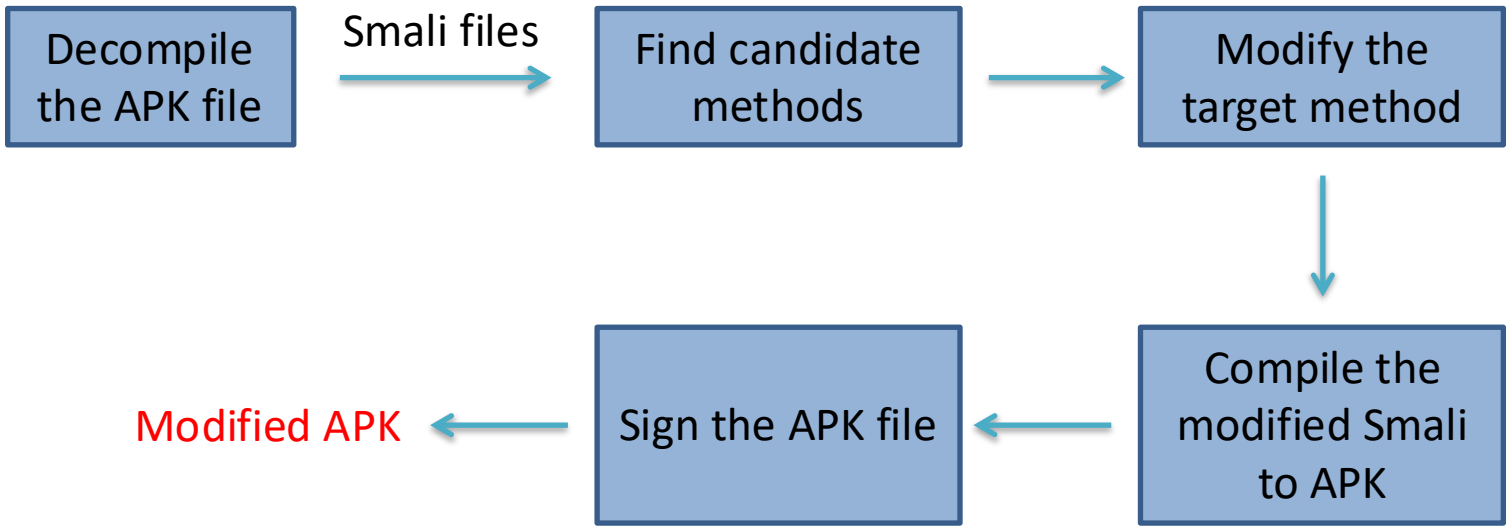
Para.	γ_3	γ_2	γ_1	γ_0
Cellular	-1.35×10^{-5}	2.9×10^{-3}	-0.21	-4.89
WiFi	-4.37×10^{-7}	-5.62×10^{-5}	-2.7×10^{-3}	0.19

If We Only Have APK Files

- APK file is an install file used in Android system
- It is not easy to get source code of most of the applications
- We develop a tool to help us modifying the third-party applications to offload version
  - Analyze and modify the APK files

APK Analysis Tool Work Flow

- We use apktool to decompile/compile the Smali code
  - Smali is the register language used in Dalvik VM
- Find methods that can be offloaded

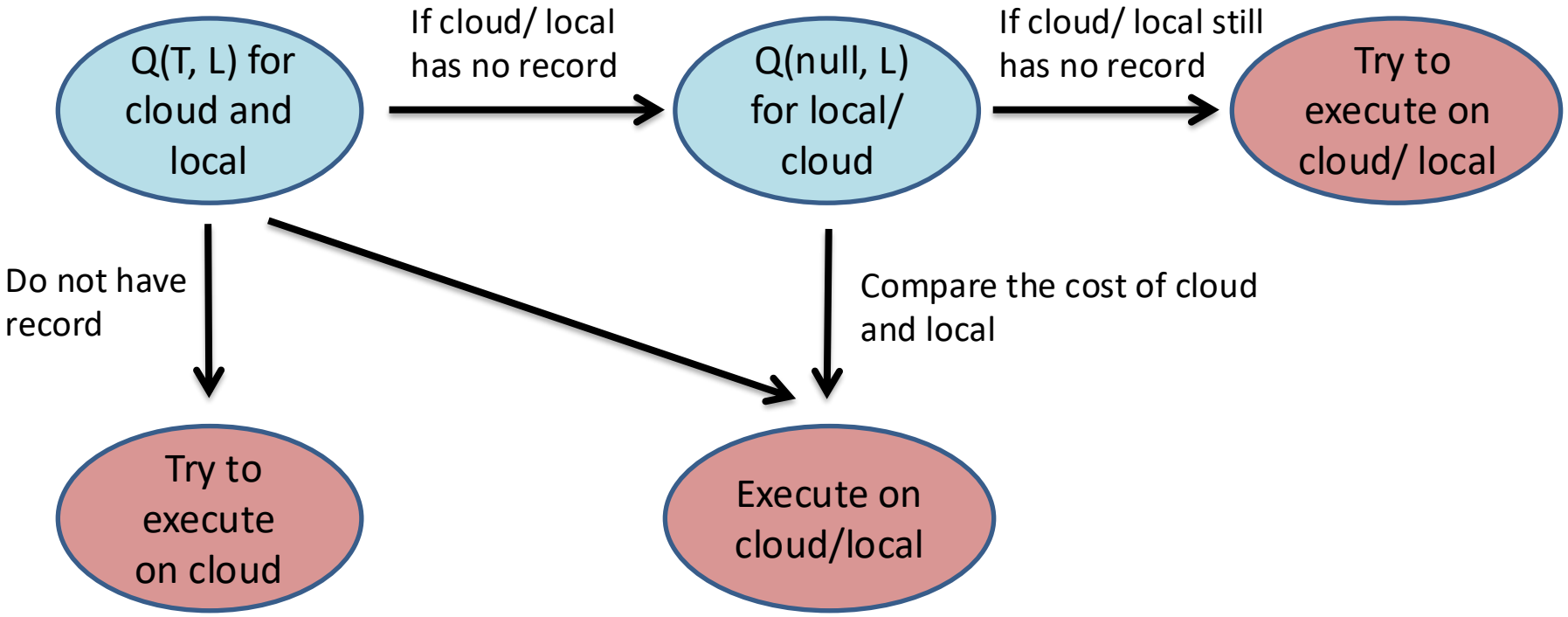


Smali Code Example

```
.class public Lcom/PinballGame/GameScreen;
.super Lcom/badlogic/gdx/InputAdapter;
.source "GameScreen.java"

# interfaces
.implements Lcom/badlogic/gdx/Screen;

# static fields
.field public static ball_1000_texture:Lcom/badlogic/gdx/graphics/g2d/TextureRegion;
.field public static ball_100_texture:Lcom/badlogic/gdx/graphics/g2d/TextureRegion;

# instance fields
.field private BALL_NUMBER:I

# direct methods
.method static constructor <clinit>()V
    .locals 5

    .prologue
    const/4 v4, 0x4

    const/4 v3, 0x2

    const/4 v2, 0x0

    const/4 v1, 0x3

    .line 233
    new-array v0, v4, [Lcom/badlogic/gdx/physics/box2d/Body;
```

Context-Aware Decision Algorithm

- Q(T, L) : query database with time T and location L for previous execution cost
  - Cost can be execution time or energy consumption



Experiment Setup

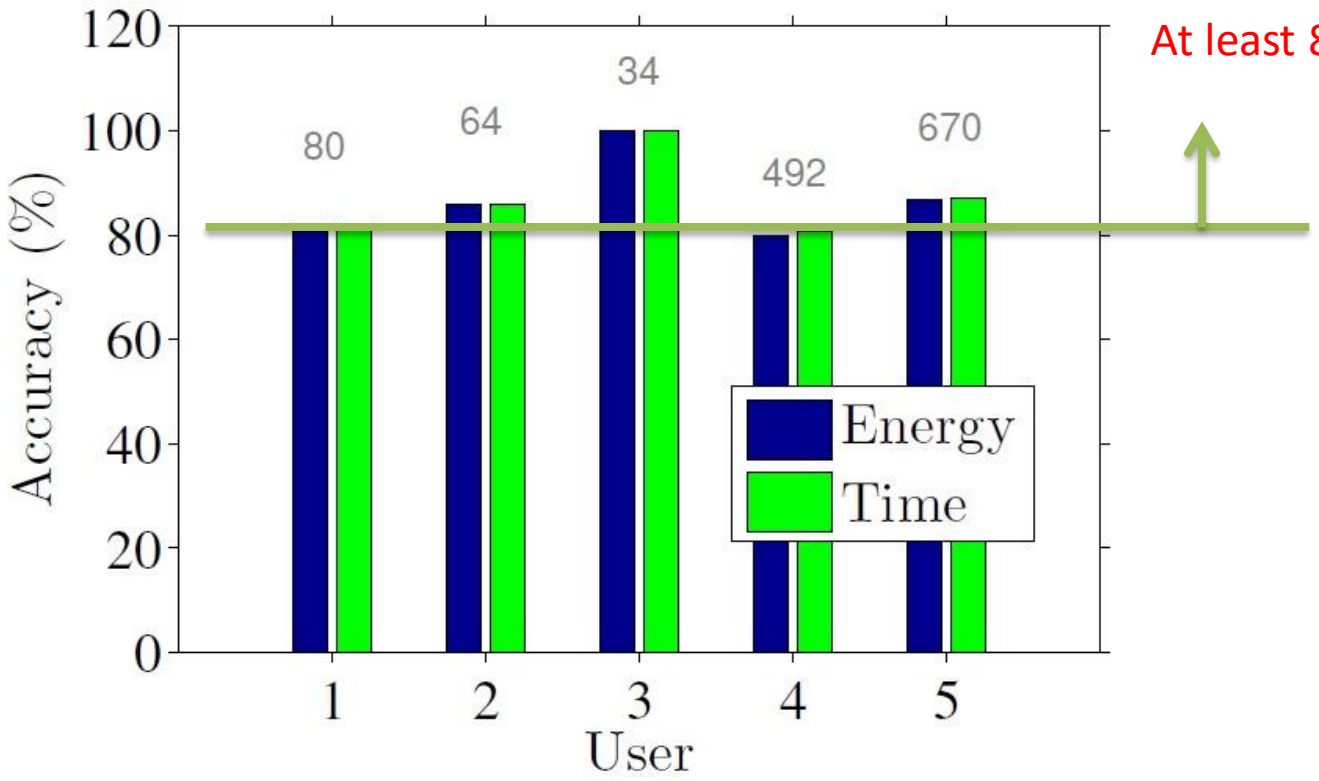
- HTC one X (client) and android X86 (server)
- We recruit 5 users to use the client in a week

Class	High Computation	Low Computation
Big State	HCBS	LCBS
Small State	HCSS	LCSS

- HCBS: image transfer and face detection
- LCBS: image transfer and color-space conversion
- HCSS: nested for-loops
- LCSS: simple for-loop
- Prediction accuracy, performance gain, overhead

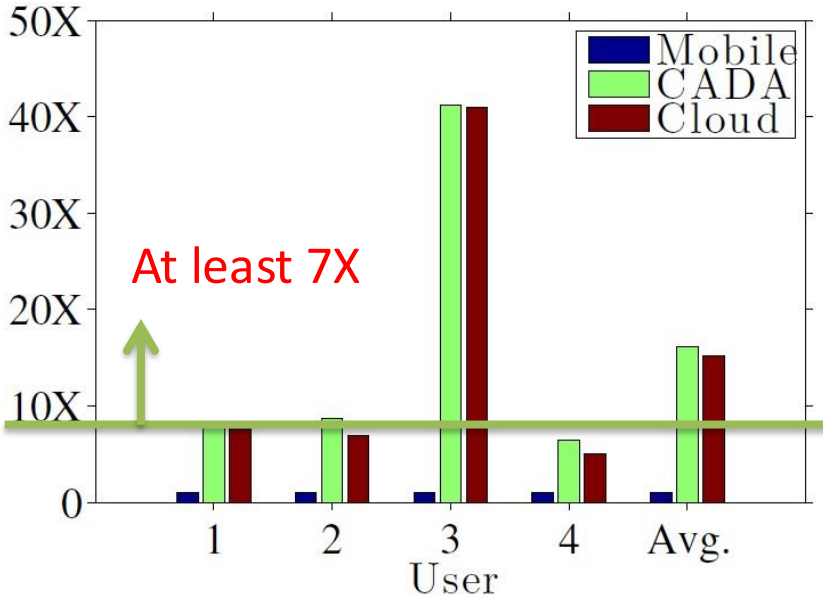
Decision Accuracy

- Take out the first two rounds which are considered as training rounds
- At least 80% across all users

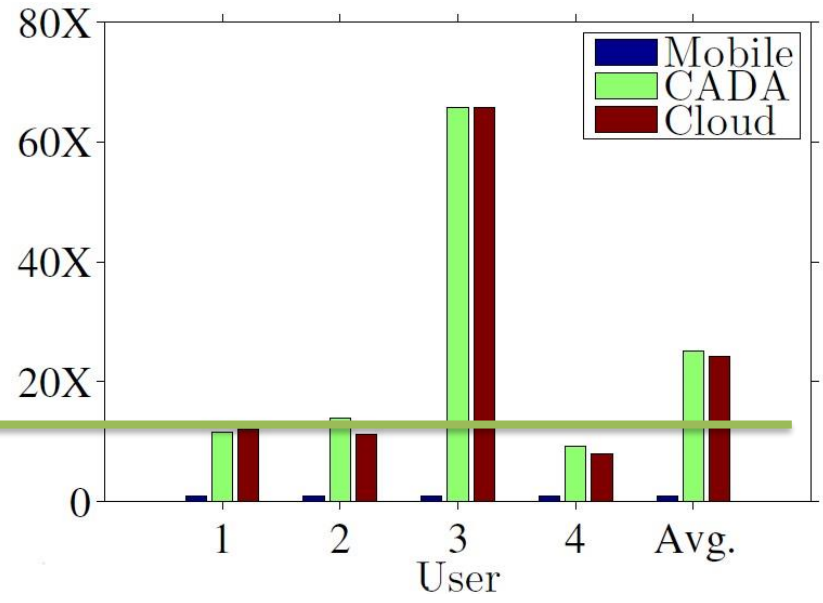


Performance Gain

- Most of the executions benefit from cloud executions, and our CADA algorithm successfully identifies the few executions which cannot benefit from cloud



Time improvement



Energy improvement

Overhead

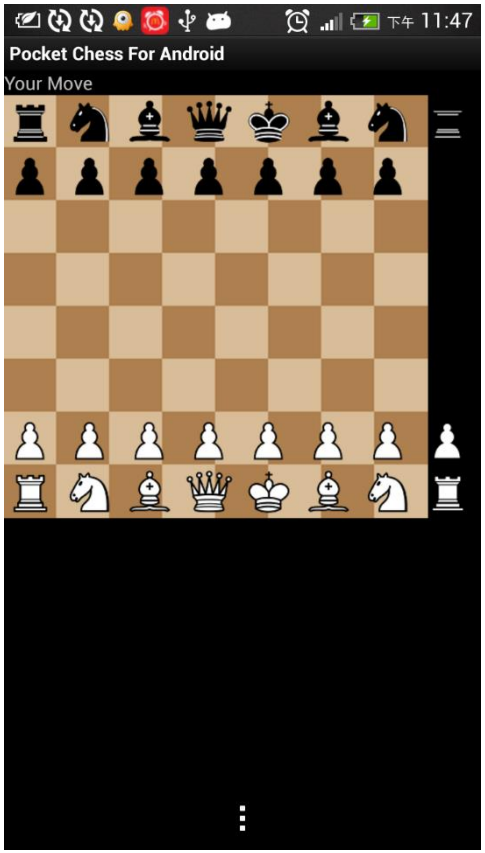
- **Space**: $M \times L \times 48 \times 7$ for each application
- **Time (runtime)**: $\text{query time} < 1$ ms
- **Energy**: About 6% energy overhead

Power (mW)	Average	Min	Max
Without profiler	48.9	47.1	50.5
With profiler	51.84	46.9	56.3

- This decision algorithm can be implemented in any offloading systems which support dynamic application offloading

Results of Offloading Third-party Applications

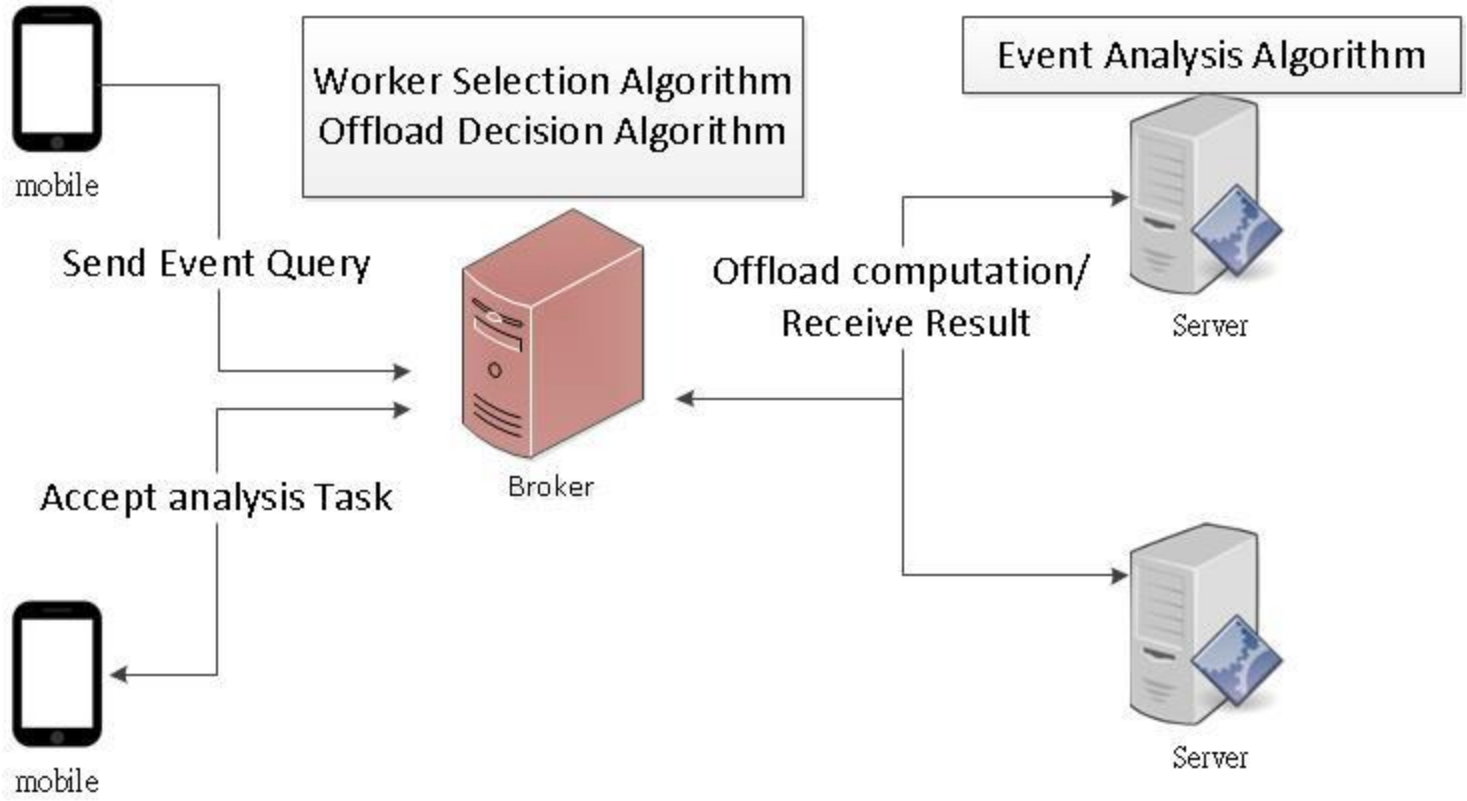
- We download four applications from Internet
- Packet Chess
  - Offload: 1742.5 ms
  - Local: 2641.5 ms



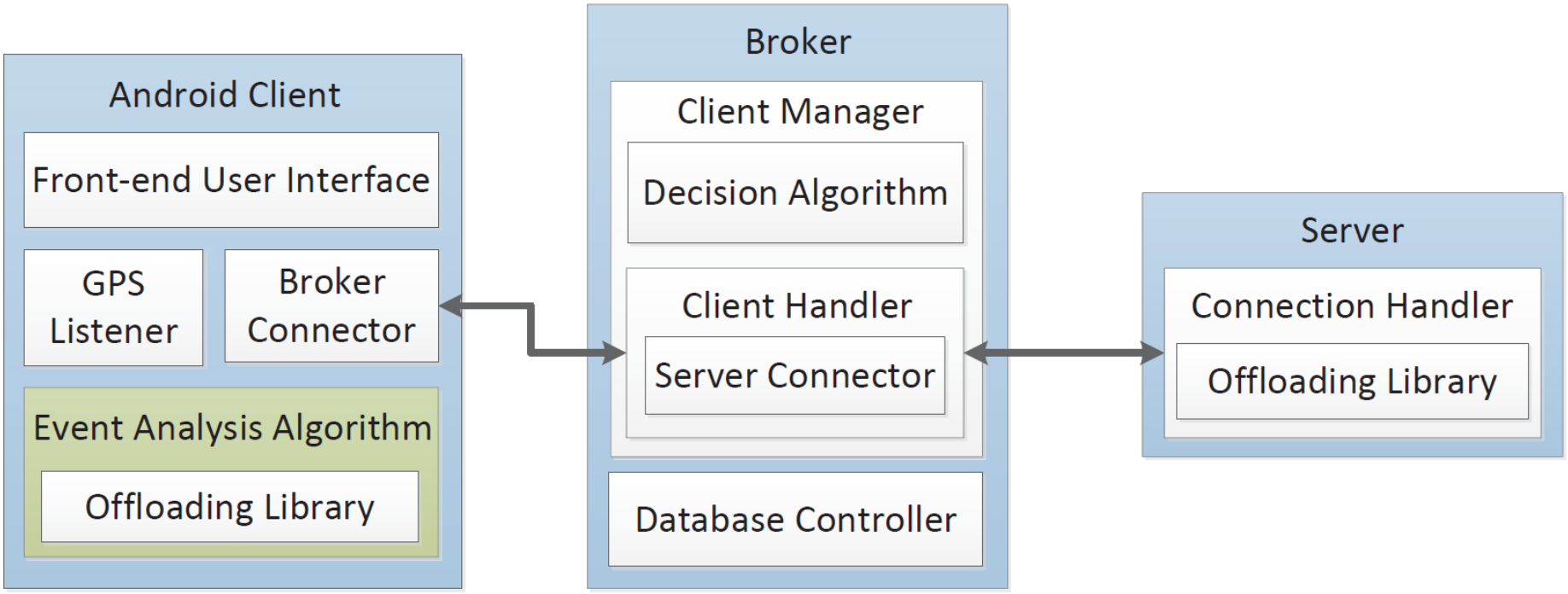
Outline

- Introduction
  - Efficient Crowdsensing System With Offloading
  - Offloading Applications To Cloud
  - Contributions
- Offloading Applications To Cloud
  - Architecture
  - Problem Statement
  - Algorithm
  - Experiments
- Offloading Event Analysis Algorithms: Using IsCrowded And IsNoisy As Case Studies
  - Implementation
  - Case Studies
  - Evaluations
- Conclusion And Future Work

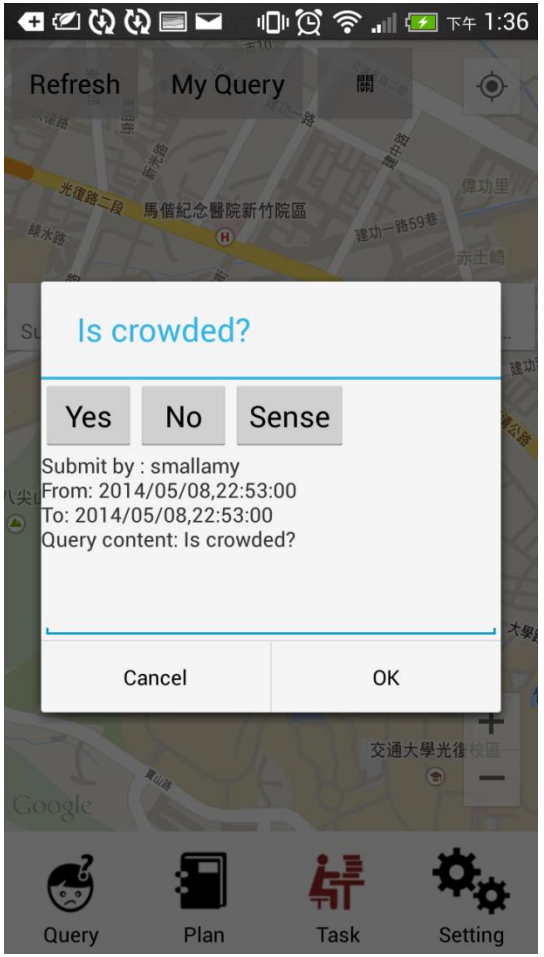
System Overview



Prototype Architecture



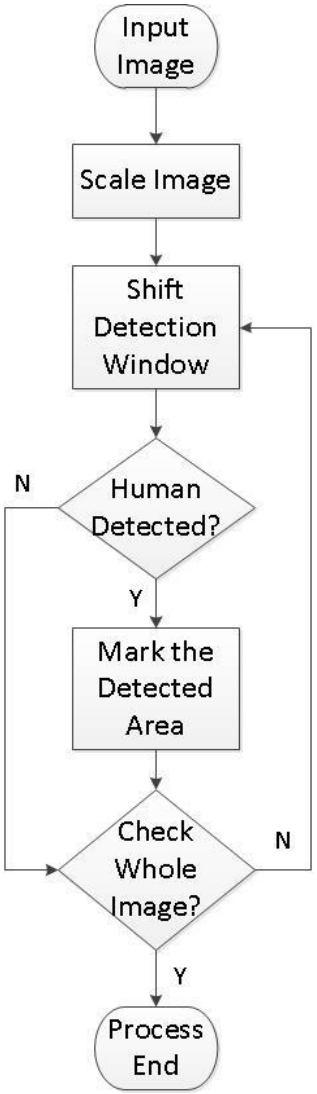
Screenshots

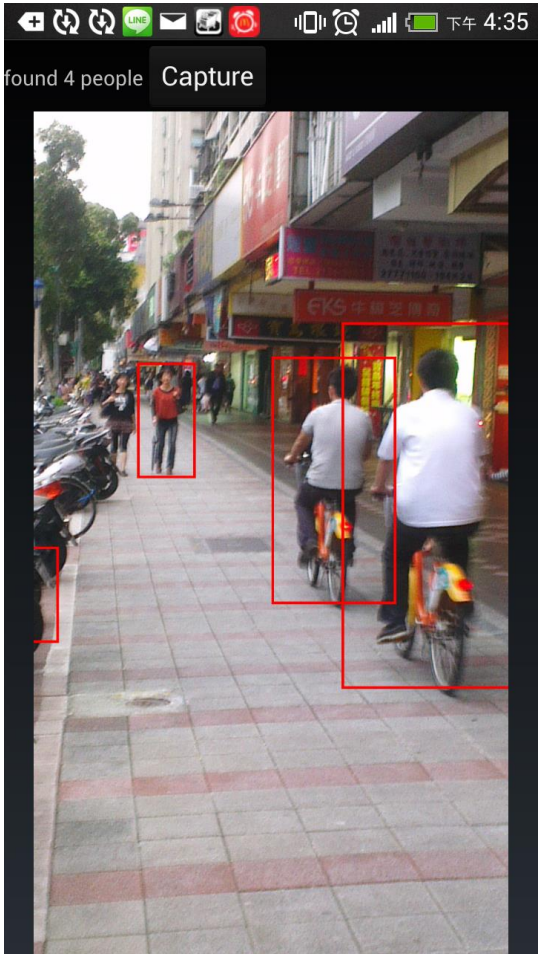


Event Analysis Algorithms

- IsCrowded
  - Human detection
  - HOG + SVM
    - Histogram of Oriented Gradients
  - SVM is trained by OpenCV
  - We use OpenCV to do the image processing
- IsNoisy
  - Record the noise by microphone
  - Compute the mean db

HOG Workflow And Example





Setup

- Broker and server
  - VMs on Desktop PC
  - OS: Android-x86
- Client
  - HTC smartphone
- Database
  - MySQL database
- We put a PC with dummynet between the broker and client
- Network delay = [0, 50, 100, 200, 400] ms
- Packet loss rate = [0, 2, 4, 8, 10] %
- Bandwidth = [256, 512, 1024, 2048, 4096] kbit/s
- We run each setting 5 times

IsCrowded (Ideal Case)

- Offloading use only 56.5% processing time and 25.9% energy consumption

IsCrowded	Min	Max	Avg.
Local execution Time (ms)	31322	33389	31957
Offload execution Time (ms)	15536	19924	18062
Local energy (mJ)	10474	10887	10618
Offload energy (mJ)	2396	2992	2759.2

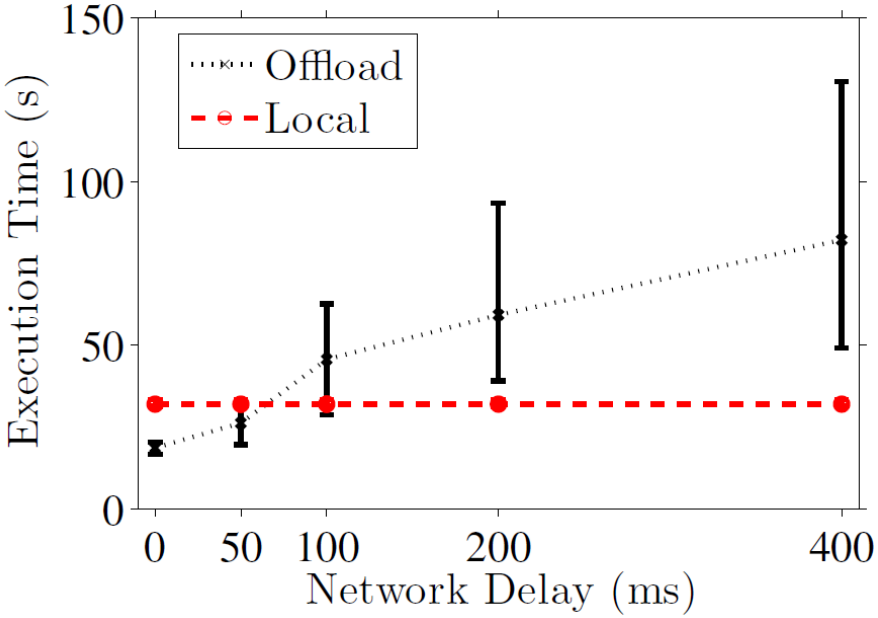
IsNoisy (Ideal Case)

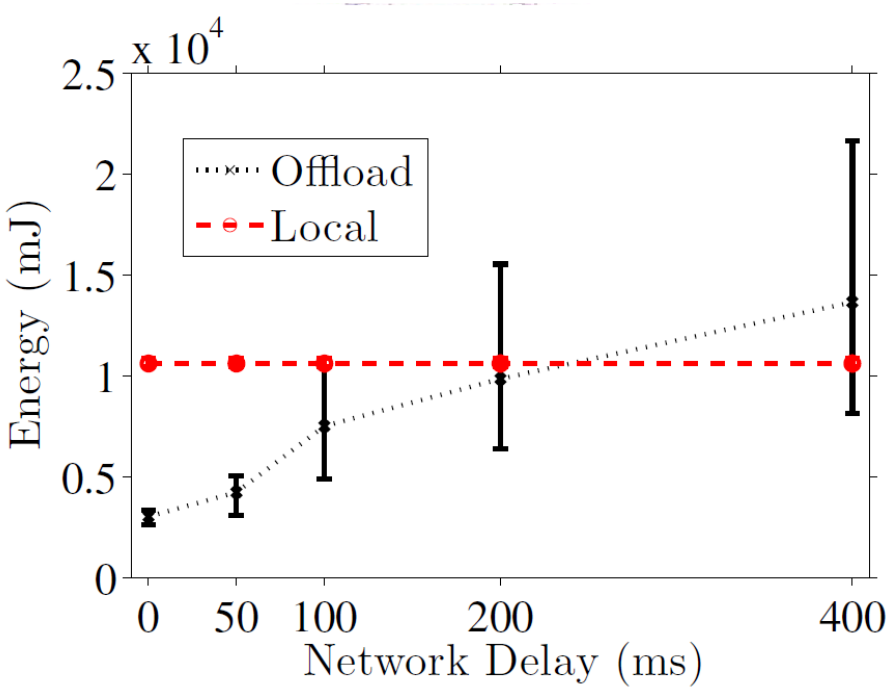
- Offloading uses 235% processing time and 184% energy consumption

IsNoisy	Min	Max	Avg.
Local execution Time (ms)	123	196	162.2
Offload execution Time (ms)	332	474	381.2
Local energy (mJ)	23	46	37
Offload energy (mJ)	54	75	68.3

Varying Network Delay

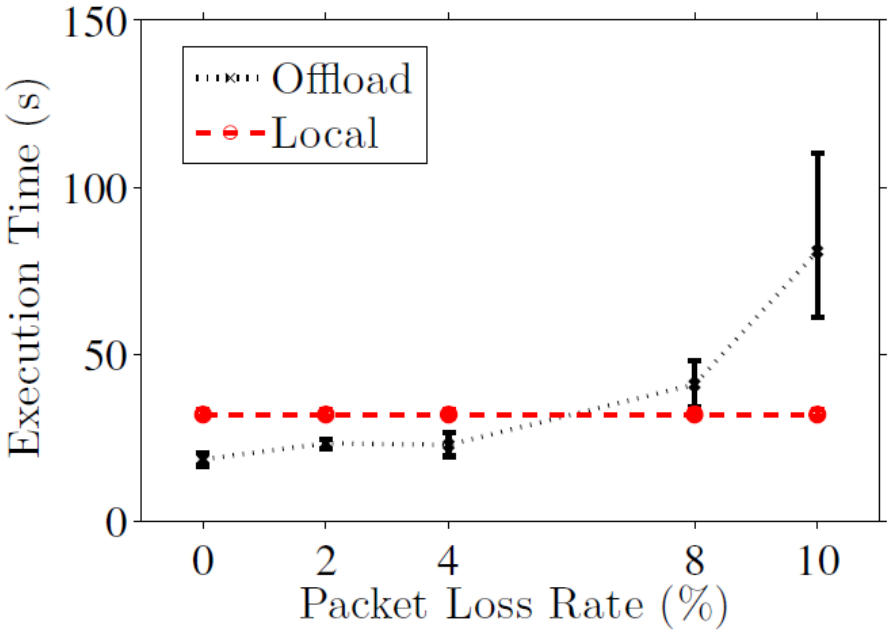
- Performance gain
  - Time: when delay $\leq$ 50 ms
  - Energy: when delay $\leq$ 200 ms

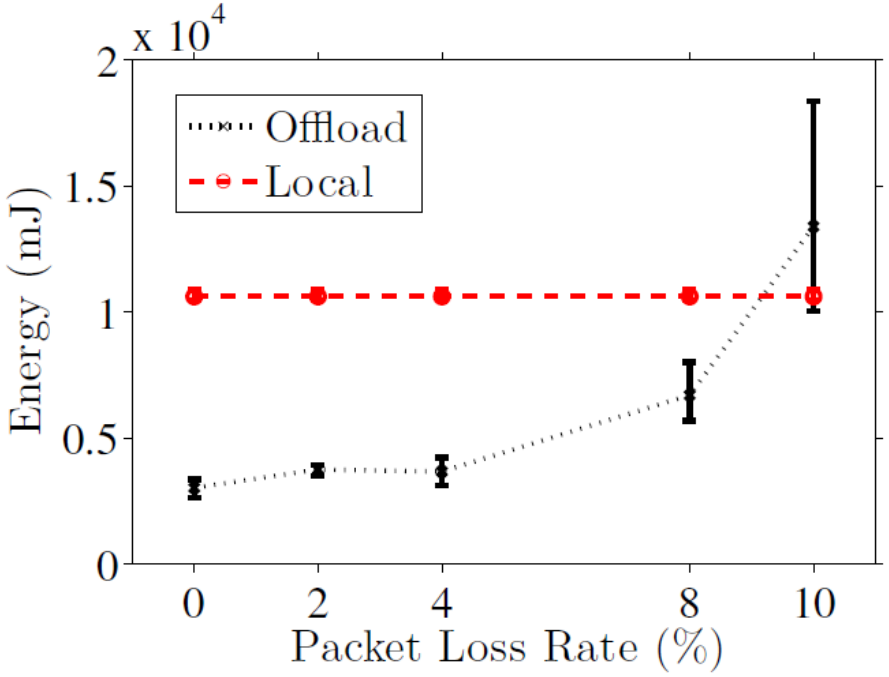




Varying Packet Loss Rate

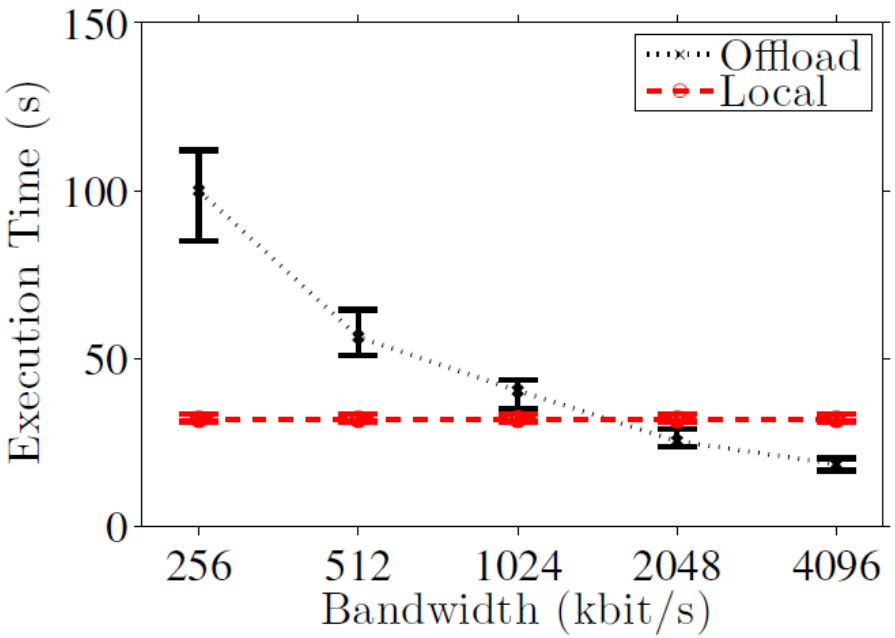
- Performance gain
  - Time: when loss rate $\leq$ 4%
  - Energy: when loss rate $\leq$ 8%

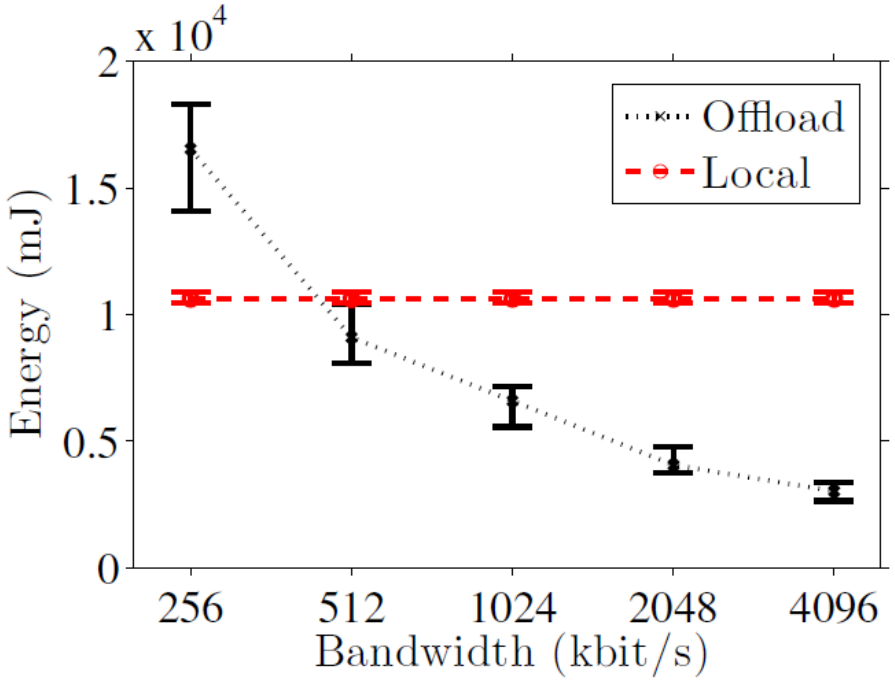




Varying Bandwidth

- Performance gain
  - Time: when bandwidth ≥ 2048 kbit/s
  - Energy: when bandwidth ≥ 512 kbit/s





Summary of The Results

- The network condition significantly affects the offloading benefit
  - Network delay: energy and time (< 50 ms), energy (< 200 ms)
  - Packet loss rate: energy and time (< 4 %), energy (< 8 %)
  - Bandwidth: energy and time (> 2048 kbit/s), energy (> 512 kbit/s)
- Offloading decision is necessary

Outline

- Introduction
  - Efficient Crowdsensing System With Offloading
  - Offloading Applications To Cloud
  - Contributions
- Offloading Applications To Cloud
  - Architecture
  - Problem Statement
  - Algorithm
  - Experiments
- Offloading Event Analysis Algorithms: Using IsCrowded And IsNoisy As Case Studies
  - Implementation
  - Case Studies
  - Evaluations
- Conclusion And Future Work

Conclusion

- We strive to minimize the energy consumption and execution time of crowdsensing systems using offloading system
- We propose a novel algorithm to determine whether to offload the computation
- The accuracy is at least 80%
- We develop an APK analysis tool to help us analyze and modify the APK files
- We offload third-party applications and show that offloading is feasible in existing applications
- We implement a crowdsensing prototype system and show that offloading improves the system performance

Future Work

- Find out other contexts that can have high accuracy for most of the mobile device users and applications
- Try to offload methods that contain local resource, for example, camera and GPS.

Thank You For Your Listening