THE LATENCY, ACCURACY, AND BATTERY (LAB) ABSTRACTION: Programmer Productivity and Energy Efficiency for Continuous Mobile Computing

Kansal et al. (2013)

November 11, 2013 **Presented by: Merhawit Habte**

1. Introduction

- Emerging mobile applications that sense context are very important; e.g. for timely news and events, health tracking, social connections, etc.
- But, continuously computing context drains mobile batteries quickly.
- Techniques available are not applicable for every scenario.
- Often developer must choose the appropriate algorithm and its parameters. This increases programming burden.
- An appropriate abstraction is required to make context sensing both efficient and widely useful.

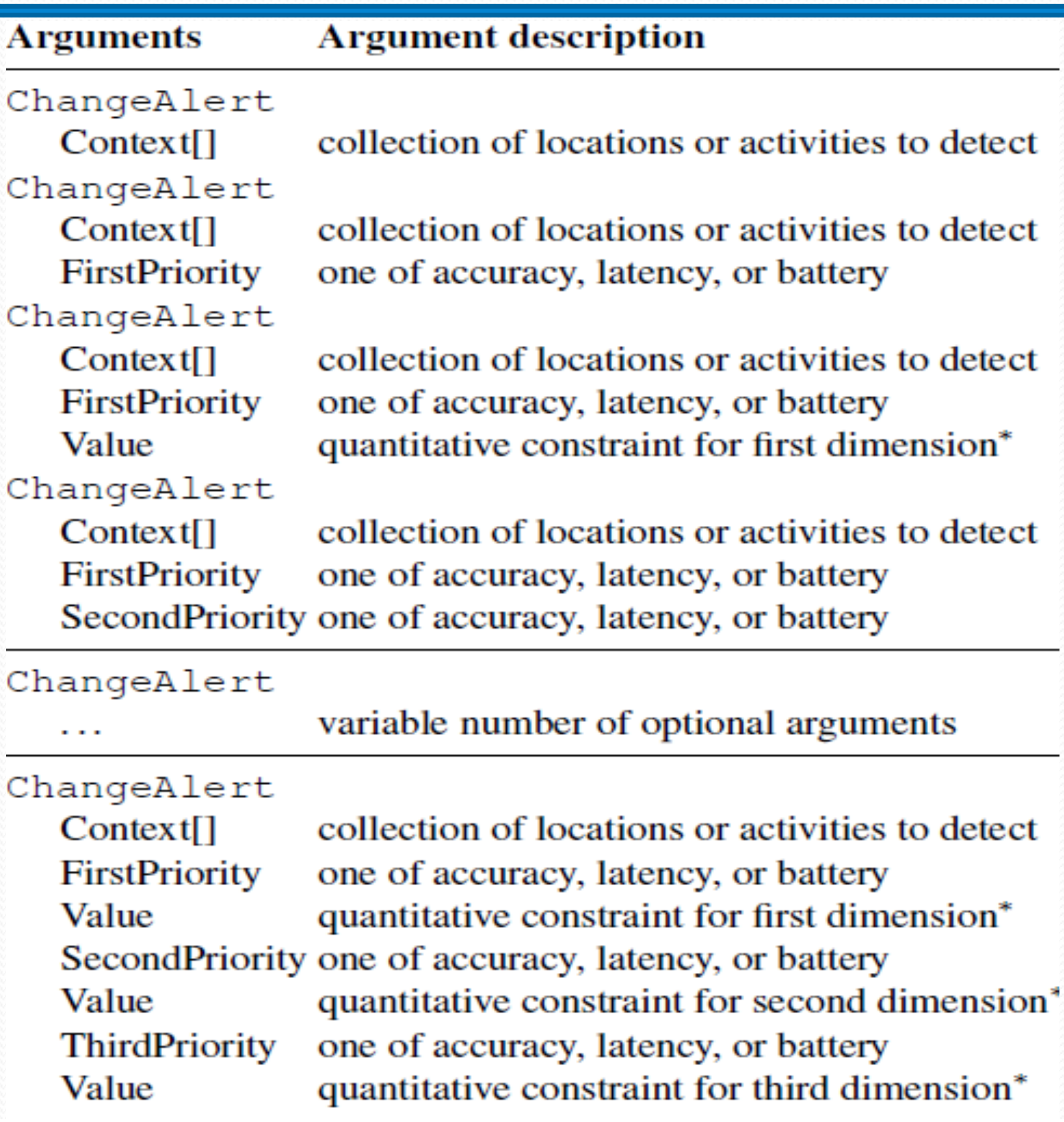
2. Objectives

- The paper shows:
  - LAB abstraction captures a wide variety of application requirements and simplifies context programming;
  - Senergy efficiently satisfies application constraints by choosing among the multiple algorithms; and
  - Senergy may optimize for multiple simultaneous applications.

3. Senergy API Design

- To increase both programmer productivity and energy efficiency, API Design should be organized based on:
  - Piority order among accuracy, latency, and energy use (battery); and
  - Partial quantitative specification of latency and/or battery consumption.
- The LAB abstraction is implemented in the Senergy API, which exposes location and activity context through an asynchronous method, `ChangeAlert`.
- The application registers a callback method, and Senergy invokes it each time Senergy detects the specified change in location or activity.
- Applications may unsubscribe with `UnsubscribeAlert`.

API and Arguments



Arguments	Argument description
`ChangeAlert`	
Context[]	collection of locations or activities to detect
`ChangeAlert`	
Context[]	collection of locations or activities to detect
FirstPriority	one of accuracy, latency, or battery
`ChangeAlert`	
Context[]	collection of locations or activities to detect
FirstPriority	one of accuracy, latency, or battery
Value	quantitative constraint for first dimension*
`ChangeAlert`	
Context[]	collection of locations or activities to detect
FirstPriority	one of accuracy, latency, or battery
SecondPriority	one of accuracy, latency, or battery
`ChangeAlert`	
...	variable number of optional arguments
`ChangeAlert`	
Context[]	collection of locations or activities to detect
FirstPriority	one of accuracy, latency, or battery
Value	quantitative constraint for first dimension*
SecondPriority	one of accuracy, latency, or battery
Value	quantitative constraint for second dimension*
ThirdPriority	one of accuracy, latency, or battery
Value	quantitative constraint for third dimension*

API and Arguments Cont…

- **First argument specifies the context, which consists of *locations* and *activities* in Senergy.**
- **Location elements are geographic coordinates.**
- **Activities are values in an enumerated type and currently include all, driving, walking, and stationary.**

  e.g., `Activity.DRIVING` **detects the beginning of driving**
  `~Activity.WALK` **detects when the user stops walking**
  `Activity.ALL` **result in a callback on all activity changes**
  `Location.ALL` **result in a callback on all location changes**
  `ChangeAlert(Location.ALL)` **continuously tracks user location**
- **Second argument (optional) specifies the highest priority choice of accuracy, latency, or energy.**
- **Third argument (optional) specifies either the next highest priority dimension or quantifies the prioritized dimension for latency or energy.**

Example API and Usage

- To detect when the user starts walking without any constraints, an application invokes:

```
Activity[] activities = {Activity.WALKING};
ChangeAlert(activities);
```

- To detect each time a user starts driving and count car trips, assuming the application can tolerate a 5 minute latency for the start driving notification, it invokes:

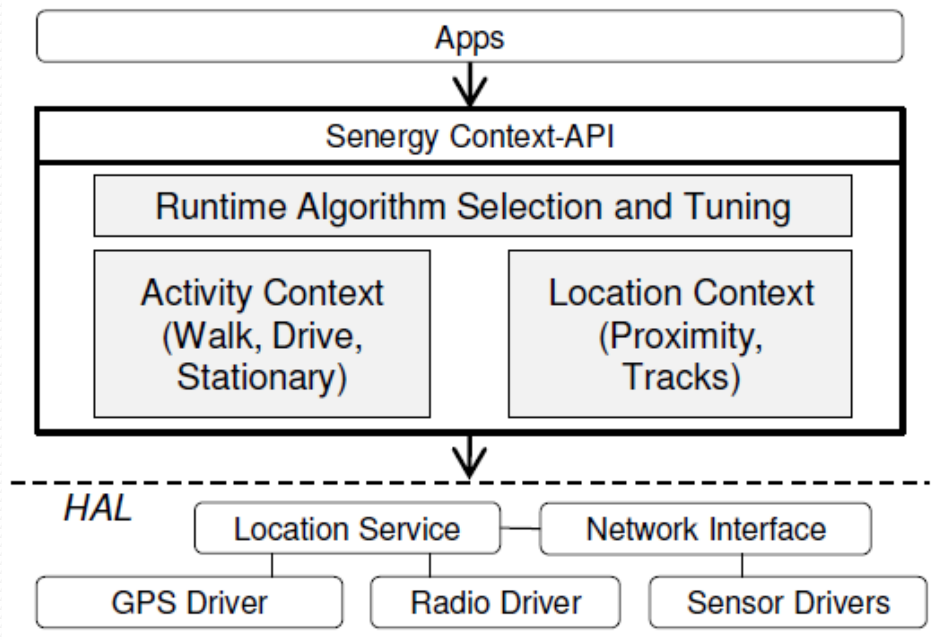
```
Activity[]acts = {Activity.DRIVING};
ChangeAlert(acts,Priority.Latency,300);
```

- Similar as above, the developer now also wants to restrict the battery impact of the application to 1% over 24 hours, the application invokes:

```
ChangeAlert(acts,Priority.BATTERY,1,
Priority.Latency,300,Priority.Accuracy);
```

4. Senergy Resource Optimization

- Senergy is implement on top of existing sensing capabilities in mobile devices
- Senergy needs multiple algorithms for each context type
- At runtime, Senergy selects the most suitable algorithms and parameter values
- The hardware absraction layer (HAL) consists of components that already exist in mobile OSs

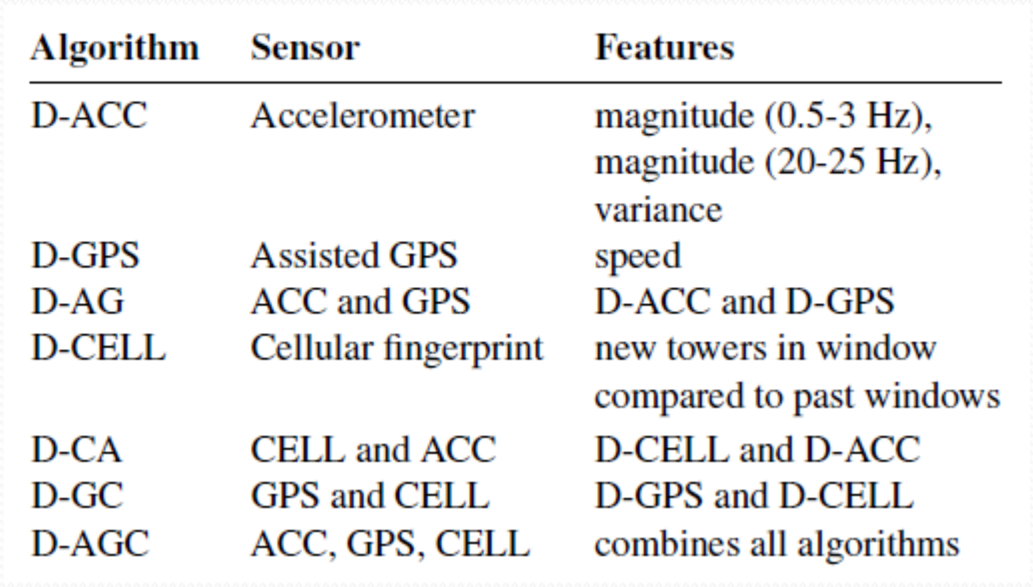

Apps
Senergy Context-API
Runtime Algorithm Selection and Tuning
Activity Context (Walk, Drive, Stationary)
Location Context (Proximity, Tracks)
HAL
Location Service
Network Interface
GPS Driver
Radio Driver
Sensor Drivers

5. Activity Context

- The following are implemented:
  - Three binary classifiers that detect the presence or absence of driving, walking, and stationary, and
  - A multi-state activity classifier that infers if the user is driving, walking, or stationary.
- Activity inference algorithms employ a typical machine learning approach:
  - sense over a time window,
  - compute features from the sensor data, and
  - classify the data using a model previously trained on ground truth.
- In Senergy a Naive Bayes classifier with supervised discretization is used.

Driving Activity

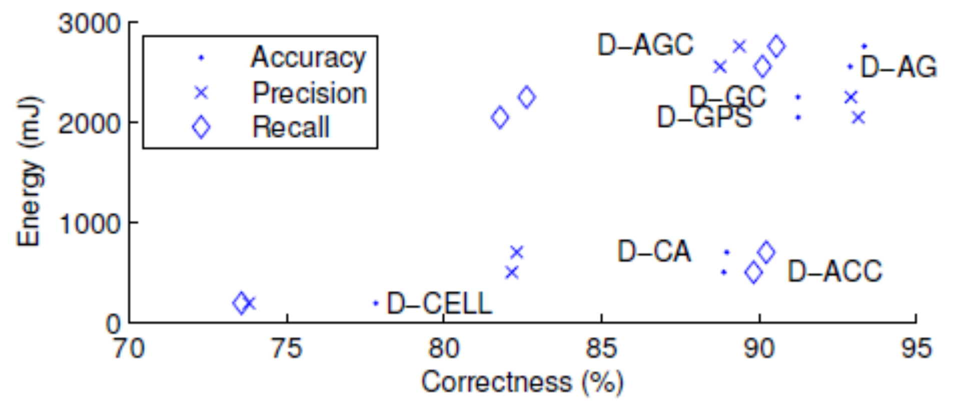
- Seven algorithms are implemented for driving detection; using the accelerometer, cellular, and GPS data
- These algorithms provide a design space that Senergy uses to tradeoff accuracy, latency, and energy.
- The tradeoffs are quantified to select the best operating point.



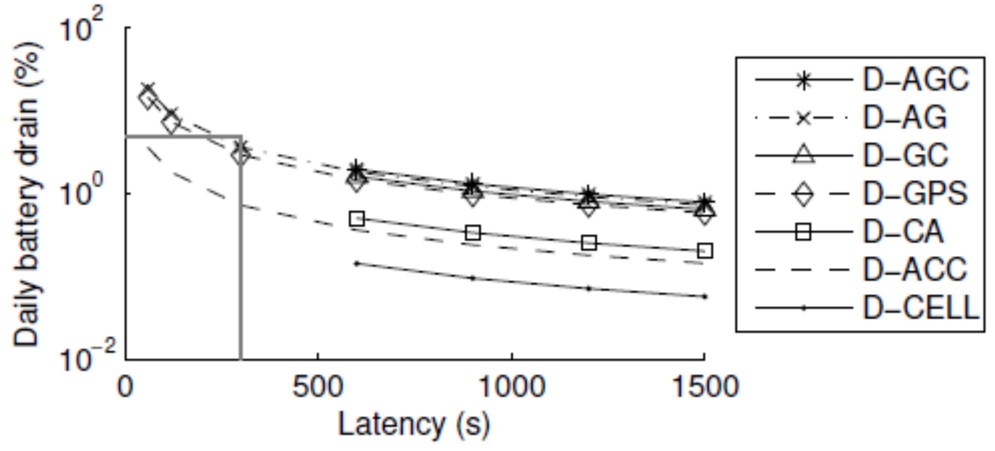
Algorithm	Sensor	Features
D-ACC	Accelerometer	magnitude (0.5-3 Hz), magnitude (20-25 Hz), variance
D-GPS	Assisted GPS	speed
D-AG	ACC and GPS	D-ACC and D-GPS
D-CELL	Cellular fingerprint	new towers in window compared to past windows
D-CA	CELL and ACC	D-CELL and D-ACC
D-GC	GPS and CELL	D-GPS and D-CELL
D-AGC	ACC, GPS, CELL	combines all algorithms

Driving Activity Cont..

- Figures shows the accuracy and latency trade-offs
- All algorithms have reasonable accuracy, precision, and recall.
- Precision indicates how many of the detected instances were indeed correct
- Recall measures how many of all true instances were detected.



(a) Accuracy



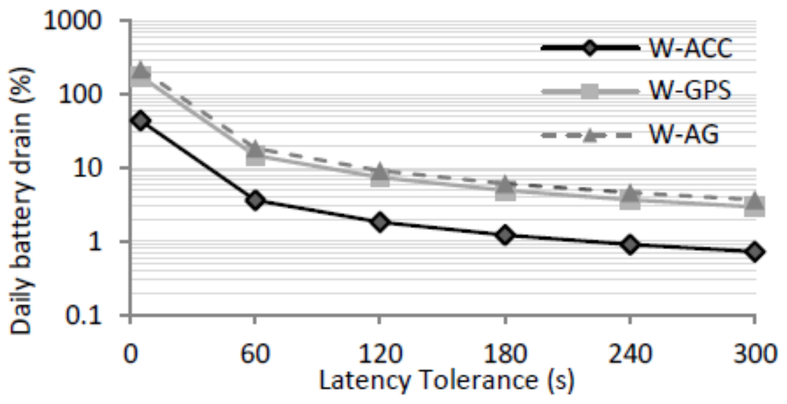
(b) Latency

Driving Activity Cont..

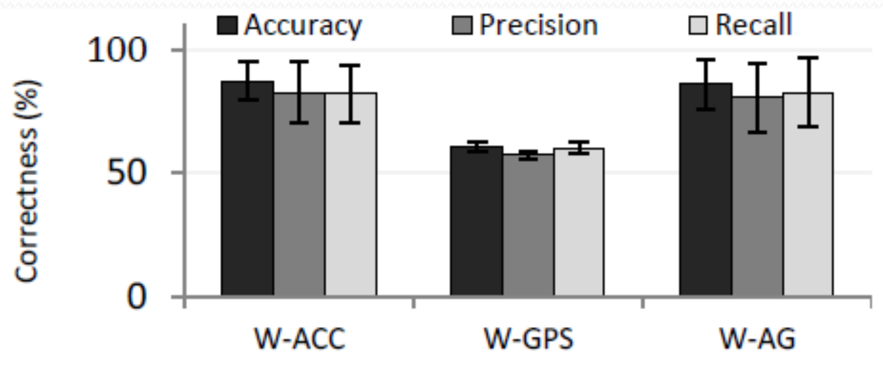
- Different algorithms may be suitable for different application requirements.
- For instance:
  - D-CELL minimizes energy if the application can tolerate a few minutes of latency.
  - D-ACC is a good candidate if the application requires low latency.
  - D-CA yields a small advantage in accuracy at a modest increase in energy but has a much higher latency.

Walking, Stationary, and All Activities

- The classifiers use accelerometer data, GPS data, and their combination.
- The following binary classification algorithms are implemented:
  - Walking: W-ACC, W-GPS, and W-AG,
  - Stationary: S-ACC, S-GPS, and S-AG, and
  - Multi-state activity: M-ACC, M-GPS, and M-AG.
- Figures below shows the accuracy and latency trade-offs with energy for walking detection.



(a) Latency and Energy



(b) Accuracy

Walking, Stationary, and All Activities Cont…

- **The algorithms that rely on GPS (W-GPS and W-AG) use significantly more energy than W-ACC, but do not always improve accuracy and hence are excluded.**
- **Figures below show the accuracy of stationary context and the multi-class detector.**
- **They consume the same amount of energy as walking detection.**

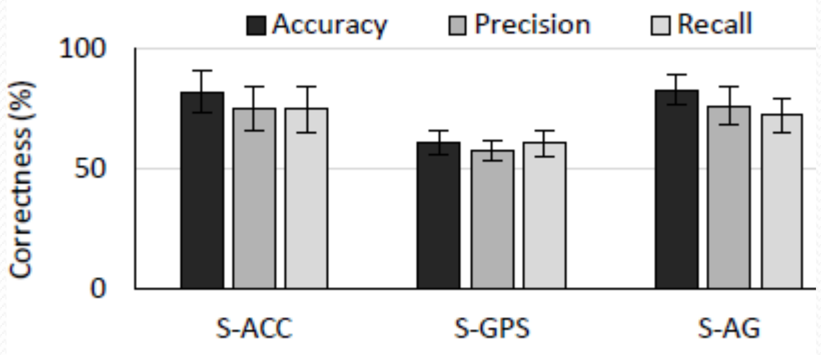


Figure 7: Accuracy for stationary activity.

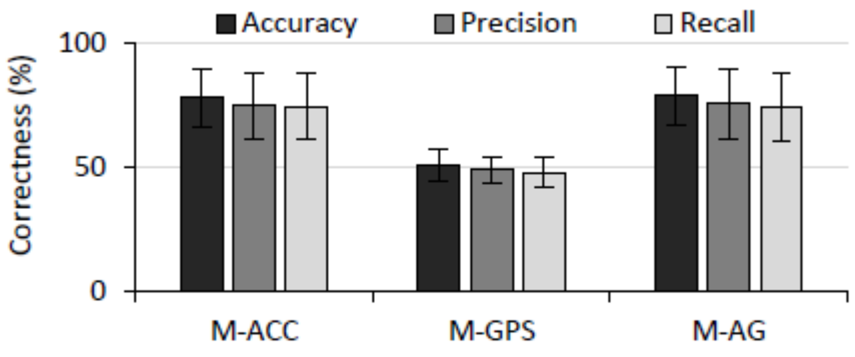


Figure 8: Accuracy for multi-class activity detection.

6. Location Context

- **Applications may invoke the Senergy API with `Location. ALL` to track all location changes or with a set of specific locations to monitor proximity to just that set.**
- **Continuous Tracking:**
  - **Senergy detects if the user is moving, using S-ACC.**
  - **The movement check interval is the smaller of the requested location update latency or 2 minutes**
  - **If the user is not stationary, location is updated.**
  - **Senergy first attempts to update location using GPS and if that fails, it uses network fingerprint based location.**

Location Context Cont..

- Location Proximity:
  - Senergy performs optimization for proximity to specific locations
  - Number of location updates is reduced based on the current distance from the nearest interesting location.
  - If the time it takes to reach a location (t_{min}) is smaller than the delay tolerated by the application, Senergy will use the tolerable delay.
  - If t_{min} is so large movement sensing is dropped until t_{min} reduces.
  - Whenever Senergy updates location, the algorithm checks if the location is within a specific radius of one or more of interesting locations.

Location Context Cont..

- Most users move less than 16% of the time during a day.
- Assuming that motion is detected every 2 minutes, Figure below shows the latency and energy tradeoff for tracking location for six different options.
- Senergy uses this analysis to select the most appropriate algorithm at runtime.
- The actual energy use for a user over a day will vary since Senergy will dynamically switch between GPS and fingerprint.

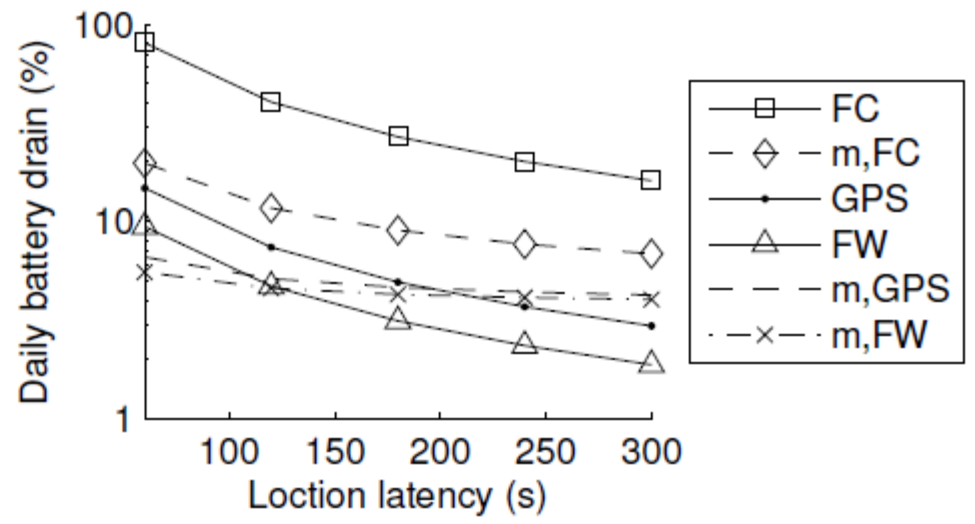
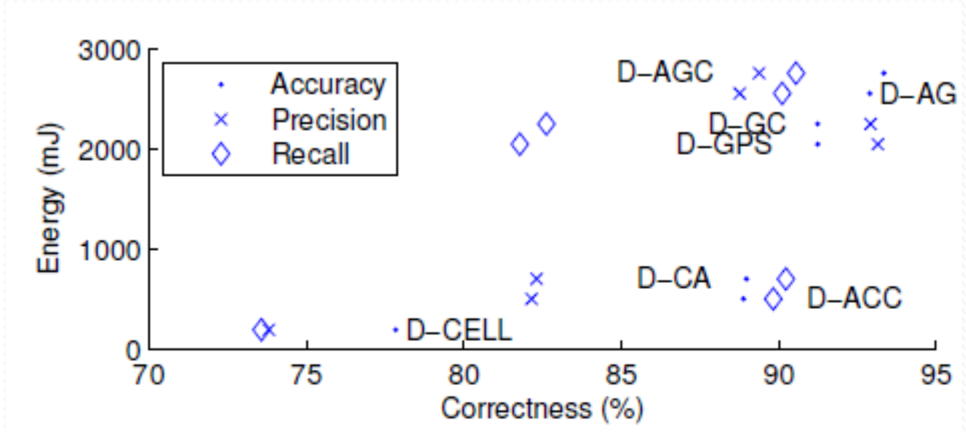


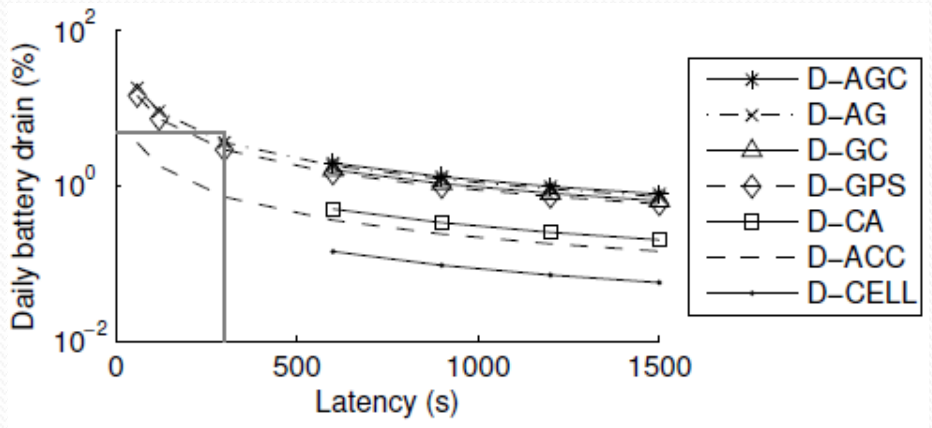
Figure 1: *Location Tracking. Latency-energy tradeoffs for six algorithms averaged over four devices.*

7. Runtime Algorithm Selection

- Algorithms that are worse on battery use are better in accuracy.
- For the same energy drain, a lower accuracy algorithm can provide better latency, and vice versa.
- Senergy selects the appropriate algorithm at runtime when an application makes an API call specifying its requirement



(a) Accuracy



(b) Latency

Runtime Algorithm Selection Cont…

- **Senergy chooses D-GA as it has the highest accuracy.**
- **Senergy defaults to maximizing energy efficiency when the API call is under-constrained**
- **Senergy would chooses D-ACC, the most battery efficient option.**
- **If multiple applications are simultaneously active, Senergy considers the constraints jointly and enforces the tightest constraints.**
  - **First it drops any battery related constraints and satisfy the accuracy and latency requirements.**
  - **If these are conflicting, latency is prioritized.**

8. Evaluation

- Senegy improves efficiency compared to the other API design to meet the requirement.
- The proposed API yields multiple orders of magnitude savings compared to the existing context API (`addProximityAlert`)
- Datasets used in Evaluation:

  1. Driving Data

     - 10 people labeled when they were driving for up to 5 days each.
     - Users tapped a button in logging application when they entered and exited their car.
     - Background collects ACC, GPS and network fingerprint scans.

Datasets used in Evaluation

2. Multiple Activity Data
   - 10 participants for driving, walking and sitting in 1 hour.
   - This data evaluted all of the activity sensing algorithms and application scenarios involving multiple activities.
3. Routine Location Data.
   - collected location and ACC data of 18 participants on their own device for 1 to 12 days.
   - they collectd data for 5 second in every minute to ensure the batteries last at least a day.
4. Workday Data:
   - They logged continous Acc data from 11 participants for 6-8 hours on one workday each
   - This data use to evaluate activity applications at extremely low latency setting.

API Configuration

1. Raw
   - They implemented simple algorithm over raw sensor data on checking location,if the user is near desired location.
   - In theory, developers could implement the best algorithms in Senergy, energy consumption is the same, unless the user executes multiple background applications where Senergy has the additional advantage of sharing context.
2. Default:
   - They emulate existing implementation in the Android OS.
   - for deleting set of locations of continuous check (Android API).
   - for those don’t have Android API, they created a default algorithm with the lowest latency implement (`addProximityAlert`).

API Configuration Cont…

3. Fixed mode
   - Implemented 3 modes prioritize energy, accuracy or latency.
4. Fixed-E
   - Prioritizes energy efficiency.
   - Uses the lowest energy inference algorithm sufficiently accurate to be included in the OS
   - E.g: location tracking use low power sensors every 2 minute and updates if the user moving.
5. Fixed-A:
   - Prioritizes accuracy.
   - It does not use low power sensor for track location.
   - Since the error in detection may miss movements and increase the overall error in location.

API Configuration Cont…

6. **Fixed-L:**
   - **prioritizes latency by supplying the lowest possible latency.**
   - **Detect movement at 5sec before sensing location.**
7. **Senegy-S:**
   - **application developer expresses one primary priority, LAB and optionally a quantitative constraint.**
8. **Senergy-M:**
   - **multiple priorities and constraint values**

Location Context Case Study

- ClubPoint
  - Senergy-S call:

```
Location[] locations = GetAAALocations();
ChangeAlert(locations,Priority.LATENCY,300)
```

Senergy-S checks for movement using the Acc and updates the location at the specified latency.

  - Senergy-M call (If the User add a 5% battery budget and request high accuracy as a third priority):

```
ChangeAlert (locations,Priority.BATTERY,5,
Priority.LATENCY,300,Priority.ACCURACY)
```

Because of 5% battery, Senergy can sense movement more frequently than the default 2 minutes

Location Context Case Study Cont..

- SimplySave: Detect proximity to participating locations with a lower latency of 60 s.
  - Senery-S call:

    ```
    ChangeAlert (locations,Priority.LATENCY,60)
    ```

    Update location and optimizes energy over accuracy by checking location if the user is mobile.
  - Senergy-M call:

```
ChangeAlert (locations,Priority.LATENCY,60,Priority.BATTERY,5)
```

- GeoReminder: Senergy only needs to detect one location with low latency.

  Update location at that latency when the user is close to the desired location.

Location Context Case Study Cont..

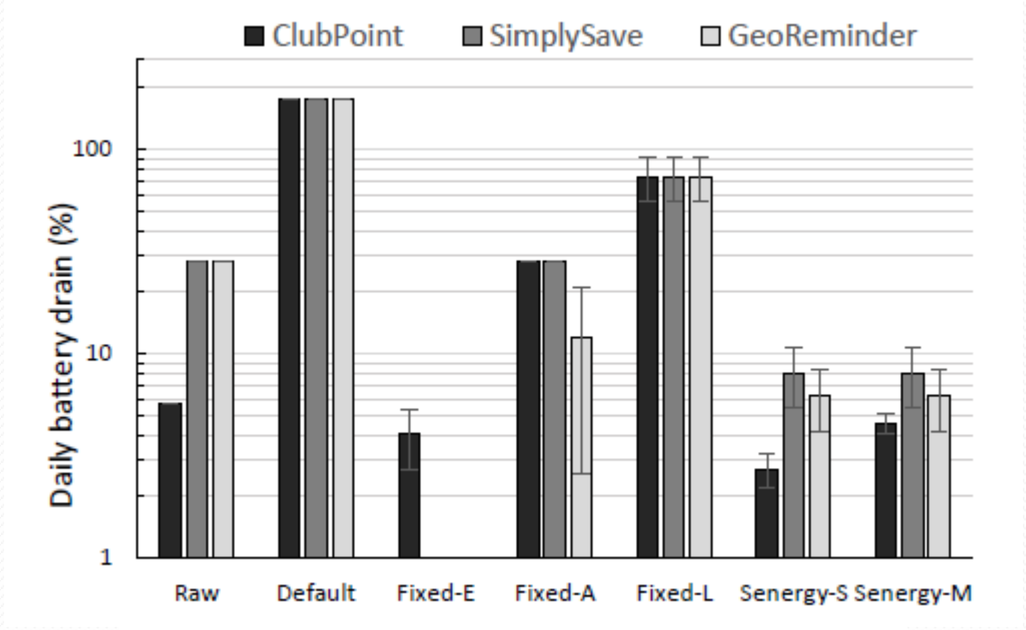


Figure 9: Location context application energy on a log scale with Routine Location Data, averaged over four platforms.

- **Values greater more than 100% indicate the battery is exhausted in less than a day.**
- **Error bars show standard deviation in behavior across 18 participants.**
- **Fixed-A, Raw, and Default do not depend on user behavior and have zero standard deviation.**

Activity Context Case Studies

- **DriverMode:**
  - **DriverMode activates a driver-mode user experience on the phone when it detects the user is in a moving vehicle**
  - **A simple Senergy-S API call is**

```
ChangeAlert(Activity.DRIVING,Priority.ACCURACY)
```

  - **Another developer may use Senergy-M specifying three constraints, prioritized: latency 60 s, 5% of the battery per day, and high accuracy as**

```
ChangeAlert(Activity.DRIVING,Priority.LATENCY,
60,Priority.BATTERY,5,Priority.ACCURACY)
```

  - **Figure compares the energy use for all API choices**
  - **Fixed-A and Senergy-S both provide high accuracy at significantly lower energy, due to increased latency**

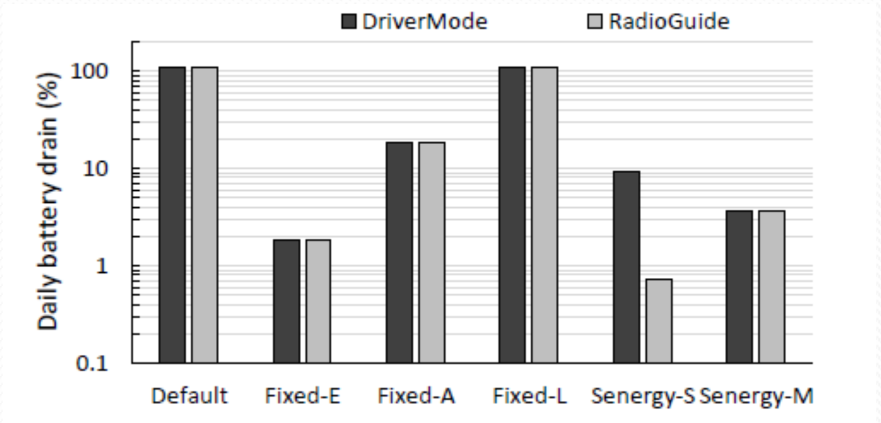


Figure 10: DriverMode and RadioGuide energy with Driving Data.

Activity Context Case Studies

- **RadioGuide:**
  - **RadioGuide publishes free local radio station schedules. Users allow the application to anonymously track when they drive and listen to the radio.**
  - **RadioGuide sets its latency to 5 minutes. The Senergy-S API call is:**

```
ChangeAlert (Activity.DRIVING,Priority.LATENCY,300)
```

  - **The Senergy-M configuration uses the same latency, restricts battery use to 5%, and requests high accuracy:**

```
ChangeAlert(Activity.DRIVING,Priority.LATENCY,
300,Priority.BATTERY,5,Priority.ACCURACY)
```

  - **Figure compares the energy use for all API choices**
  - **Senergy-M does use less energy than the other APIs, but more than Senergy-S, since it improves accuracy**

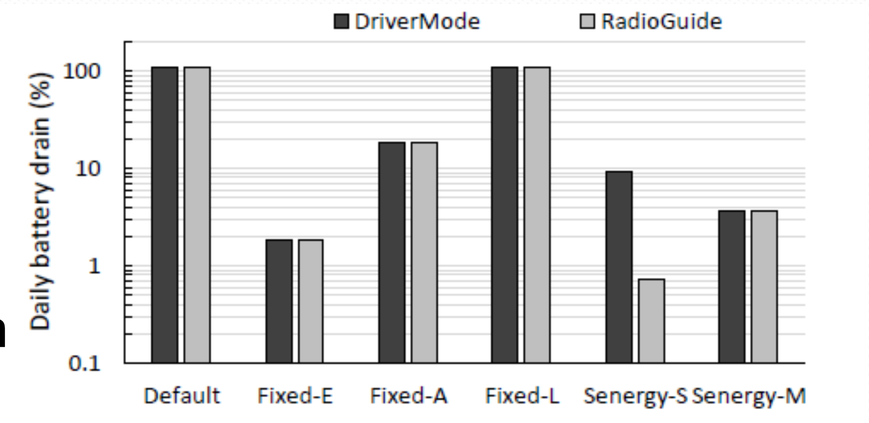


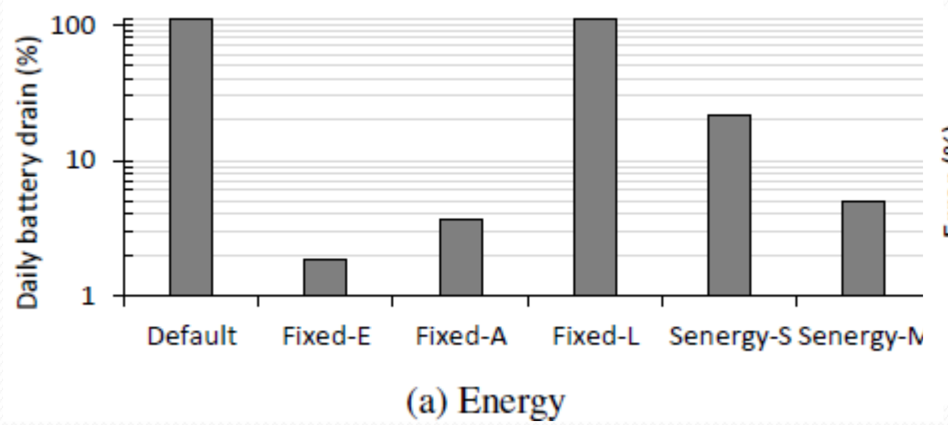
Figure 10: DriverMode and RadioGuide energy with Driving Data.

Activity Context Case Studies Cont..

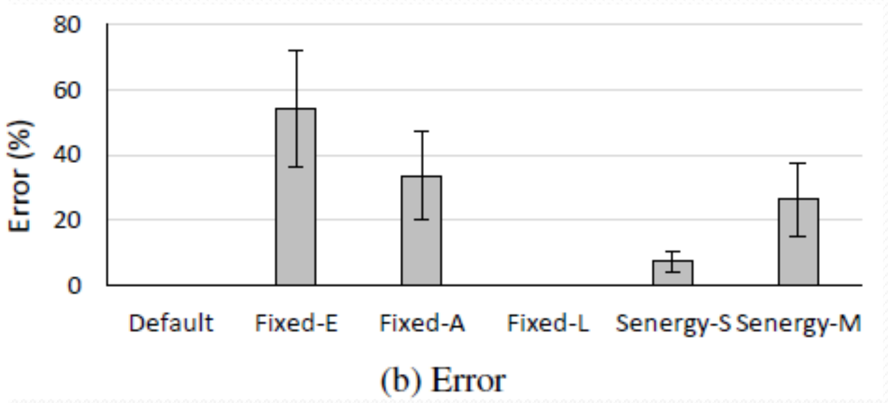
- **FitnessTracker**
  - **FitnessTracker counts a user's daily steps to estimate calorie use.**
  - **FitnessTracker wants a callback whenever the OS detects walking.**
  - **To not miss short walks, the developer requests a 10 s latency with the Senergy-S API call:**

    ```
    ChangeAlert(Activity.WALKING,Priority.LATENCY,10)
    ```
  - **To control battery use, Senergy-M call includes a 5% battery limit as the first priority constraint:**

    ```
    ChangeAlert(Activity.WALKING, Priority.BATTERY,5, Priority.
      LATENCY,10,Priority.ACCURACY)
    ```



(a) Energy



(b) Error

Figure 11: FitnessTracker energy and accuracy trade-offs using Workday Location Data.

Multiple Simultaneous Applications

- **With Senergy, energy is spent on the sensors once and the sensor data is used to compute all context outputs needed.**
- **If the algorithm used for one application suffices for others, Senergy does not use other sensors or algorithm.**
- **If one algorithm senses with more accuracy or lower latency, other applications benefit from it.**

- **Loc. apps consists of all three location apps. executing at the same time**
- **Act. apps consists of the three activity context based apps.**
- **All apps denotes all six of these applications.**
- **Acc.Loc is the forth app**

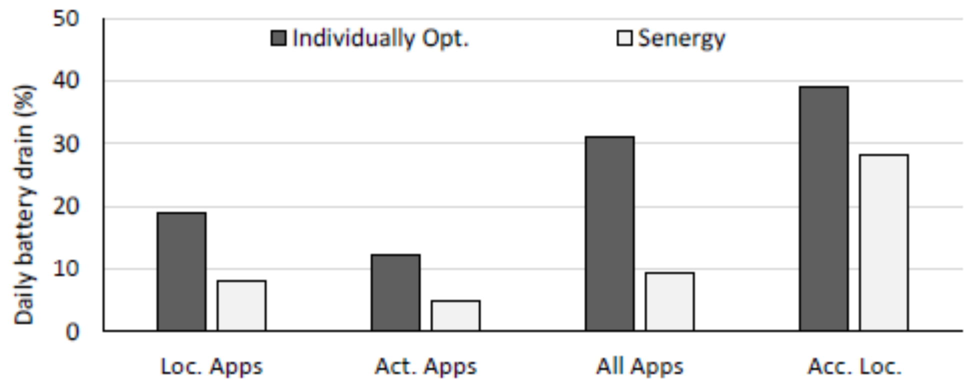


Figure 12: Energy savings when individually optimizing each application and using Senergy.

9. Discussions

- Hardware Architectures
  - New hardware broadens the range of choices available to Senergy and it may consequently satisfy more application constraints.
  - The tradeoff space should be characterized to include these options. The API does not change, but applications benefit from hardware advances.
  - If hardware advances make new types of context and activities feasible the API does not change but starts delivering these additional context types to applications.
- Predictive and Historic Context
  - Senergy does not maintain historical state.
  - If Senergy records context, it could learn user behavior models and use them to optimize context sensing.

Discussions Cont..

- Reporting Latency
  - Latency is the delay in detecting and reporting a context change to the application.
  - Additional optimization opportunities arise by decoupling the detection and reporting latency.
  - The API would evolve to add a reporting latency.
- Privacy
  - Any privacy related parameters have not been included in the API design.
  - When Senergy operates in a battery efficient mode that satisfies multiple simultaneous applications, the accuracy achieved may not be appropriate for all applications.

10. Conclusion

- This work contributed the following:
  - The paper identified the LAB abstraction and showed how to implement energy efficient continuous context sensing and how it improves programmer productivity.
  - The paper described a prototype implementation using 22 activity and location tracking algorithms.
  - The paper illustrated how the Senergy runtime uses the LAB requirements specified by applications and the algorithm tradeoffs to deliver energy efficient context sensing under a wide variety of accuracy and latency requirements.
  - It was shown for six realistic applications, how Senergy uses a small amount of application flexibility to reduce the battery drain to much more practical levels, compared to using existing APIs.