Rainier: A Cloud based Programming Framework for Collaborative Sensing Applications

Victor Bahl
Microsoft Research

Aman Kansal
Microsoft Research

Romit Roy Choudhury
Duke University

ABSTRACT
We envision a future where sensing capabilities are incorporated into a large number of commercial and social systems. Such a future may not be too distant – mobile phones are already equipped with a variety of sensors, while newer home appliances, offices, and automobiles are beginning to adopt similar trends. We believe that such proliferation of sensors lends itself to a new class of powerful, collaborative sensing applications. In an attempt to catalyze these possibilities, we observe the need for a software development environment that can natively support platform heterogeneity and collaboration. We propose Rainier, a programming framework in the cloud for collaborative sensing applications over heterogeneous devices. Rainier’s core philosophy lies in decoupling platform-independent application logic from platform-dependent sensing. By exporting the logic to the cloud (with virtual access to the sensors), Rainier creates a single software development environment that can dynamically harness scattered sensors in the wild.

1. INTRODUCTION
The past decade has seen rising popularity in programmable sensing and monitoring. The research community has focussed on wireless sensors networks where multiple sensors are combined with a microcontroller and a wireless networking interface, together called motes. The motes are typically deployed/scattered at the site of interest; they are programmed to measure specific occurrences, compute pre-specified tasks, and collaboratively transmit back answers to a remote base station. This model has found a variety of domain-specific applications, such as in volcano and environment monitoring [9, 6, 12], animal and sea-creature tracking [10], data center heat mapping [], health monitoring [4, 11, 7], asset tracking [5], etc. Authors in [1] present an interesting survey of programmable sensor networks, and their influence of different scientific disciplines (such as geology, zoology, ocean sciences, etc.).

The sensor networks described above may be classified as special purpose, vertical systems. These systems are tailor-made for specific applications, and deployed/operated/maintained by a team of experts. In contrast, we notice emerging trends in general purpose sensing, where rich sensing platforms are developing without a necessary binding to a specific application. Mobile phones are an example of such trends. The availability of a rich set of sensors, supplemented with other advantages (e.g., always-on, human-supervised, periodically-recharged), makes mobile phones a fertile ground for application development. Efforts in this direction have demonstrated early success through the app store model. However, deficiencies exist in the path of promoting the applications to a greater level of richness and sophistication. We briefly discuss this next.

We observe that there has been tremendous interest in community-based applications. Characterizing traffic congestion on roads, learning the social events around a place, meeting people with shared interest, crowdsourcing, are only a few of these nascent applications. These applications are different from today’s “personal apps” – they rely on community-wide collaborative sensing across multiple sensors and devices, as well as field data to often train their applications. We argue that today’s app-development tools are inadequate to meet these requirements. The inadequacies arise from:

1. The lack of native support for heterogeneous devices.
2. Limited programmability and libraries to manage scattered sensors.
3. Inability for a single person or a small business to gather and manage field data, often necessary for training/calibrating the app.
4. Network cost of coordination between sensors and web services.
5. Limited CPU resources on-phone for sophisticated operations.

In an attempt to address these problems (and reduce the barrier to entry for developing collaborative applications), we envision Rainier, a cloud based programming framework for virtual sensing and computing over heterogeneous devices. Figure 1 summarizes Rainier’s departure from conventional programming models, i.e., while today’s application programmers program
Figure 1: Departing from the conventional programming framework with Rainier. (a) Today, the app developer programs the phone, which in turn offloads operations to the cloud and web-services. (b) The Rainier model allows programmers to develop apps in the cloud, using the phones as the backend.

the phone with the cloud in the backend [3, 8], we propose the converse. We envision that phones (and their embedded sensors) can become the backend, while application developers can remain platform agnostic, and focus only on programming in a single cloud-based environment. Advantages of this model include availability of a rich sensing and programming library, sharable code and data, reduction of network traffic, greater CPU power, and room for optimization. Of course, challenges exist, especially in terms of software design, scalability, coping with sensor faults and uncertainty, energy, privacy, etc. Rainier is targeted to address these challenges and facilitate a cloud based programming environment.

2. MOTIVATION AND OVERVIEW

A natural question is why is the cloud necessary to develop the envisioned class of applications? As mentioned earlier, Rainier is targeted to enable apps that are difficult to develop today. These difficulties arise from one or more of the following reasons.

- With advances in data mining, machine learning, natural language processing, signal processing, and multi-dimensional sensing, a variety of applications can be enabled. However, such applications typically require far more CPU power than what may be available to phones. Exporting such compute-heavy tasks to the cloud appears to be a natural design decision.

- Apps often collect field data for training/calibrating their algorithms. However, the scale of the data is typically small, curbing the efficiency (or even correctness) of the algorithms. The cloud provides a common plane on which the gathered data from multiple apps may be merged. The expectation is that this merged data is likely to enable even new apps that would otherwise be infeasible.

- While most of today’s apps are “personal”, community based collaborative sensing apps of the future would be difficult to achieve through P2P mechanisms. The cloud provides a rendezvous point for developing collaborative applications that engage multiple, scattered devices.

- The Microsoft programming environment (e.g., Visual Studio) offers a rich set of functionalities – phones provide a far smaller subset of these. Programming inside the Rainier cloud allows the programmer to be unconstrained by the phone limitations. The programmer develops applications as if she/she is working on a windows desktop environment; the developed code automatically coordinates mobile phones to compute the answer.

- When an app access multiple web services (such as location, mapping, notification), or certain databases (such as Flickr pictures), the energy and bandwidth costs can be excessive. Programming in the cloud executes these services over the wired network – the wireless service provider is relieved of excess demand and signaling. In view of the growing density of users, such traffic isolation (behind the cloud) adds to the motivation for using a cloud. Moreover, when all the services are running within the same data center (such as in Microsoft’s cloud), the cost of Internet traffic is also bypassed.

Towards providing this unified programming environment, Rainier integrates multiple software components. Briefly, computation is decoupled from the mobile device platform and exported to the cloud – the mobile only supports a sensor abstraction layer (SAL) that presents a high level API to access physical sensors. This isolation of computation facilitates a single programming environment in the cloud, with a variety of
mobile phones scattered in the backend. To lower the barrier of programmability, Rainier provides a library of virtual sensors (VS), defined as a logical aggregation of sensors and actuators, glued with software logic. An examples of such a VS is an “indoor/outdoor” sensor or a “car driver” sensor. Such sensors do not exist in physical reality, but are composed of physical sensors (such as light sensor, accelerometer, compass) and programmer-specified logic. A library of such sensors in the cloud enables easier application development, while facilitating collaboration among app developers. The developer of a location-based advertisement app may leverage the indoor/outdoor app, developed by an expert on indoor/outdoor sensing.

Various apps and VSs may also rely on field data to train/calibrate their algorithms (e.g., WiFi based localization requires wide-scale war-driving data). Collecting such data often raises the barrier to entry for developing richer applications. Isolated projects often war-drive small areas towards demonstrating proof-of-concepts, however, these prototypes cannot evaluate systems at scale. Rainier will provide a data plane that assimilates limited field data from different applications and “merges” them effectively to benefit each app. In fact, careful merging may enable new apps that were otherwise infeasible from the isolated data. Finally, Rainier also provides native support for collaboration among apps, with optimizations for energy and bandwidth. A traffic congestion app may require accelerometer readings from at least N sensors on a given road; Rainier will be able to pick the N devices appropriately, based on devices that are already on, have good network connectivity, higher energy levels, and other utility functions. In sum, Rainier provides a consolidated programming framework in the cloud for developing new classes of collaborative, sophisticated, data-driven applications.

3. APPLICATIONS

We envision a variety of applications feasible on Rainier. Besides motivating the platform better, we believe that these applications will also justify some of the design decisions we have made in Rainier’s architecture. We begin with the simpler apps and describe the sophisticated ones thereafter.

TrafficCongestion

Understanding the traffic congestion (and hence, commute time) along a road/highway is of interest to many services. Developing this app on Rainier will be easy. The app developer would be able to query the accelerometer readings from mobile phones on a specified road and infer the congestion state. Rainier will transparently recruit mobile phones from that road, query them in a platform-specific manner, eliminate outliers, and return the sanitized/aggregated readings. The cloud programmer remains completely agnostic of these underlying operations, and only implements the logic. With such a support for crowd-sourcing, various other applications can be developed, including BusTimes (real time bus timings), ParkNet (finding parking spots), CrowdCam (asking people for pictures/video of a place), SoundPollute (mapping sound pollution in towns/cities), etc.

PathFinder

Finding a person in a public place, such as in a library, conference hotel, or shopping mall, can be difficult. The difficulty arises from not knowing where the person may be at that time; even if known, navigating through an unfamiliar place may be frustrating. Nevertheless, mobile phone sensors (accelerometer and compass) and opportunistic user-encounters (via bluetooth) may offer opportunities for developing an electronic escort service. By periodically learning the walking tracks of different individuals, and “stitching” them carefully in the cloud, a path can be computed between any pair of persons (say Alice and Bob). Thus, Alice can be shown an arrow on her phone; so long as she follows that arrow, she is expected to reach the location of Bob. Rainier app developers should be able to develop such an application with relatively less effort. This will be particularly true, because Rainier will provide rich libraries that translate noisy accelerometer and sensor readings to “tracks” on which individuals have walked. Rainier will also provide the intersection points—the final output will appear as a graph. The app developer will only need to implement his choice of routing algorithm on this graph. Again, the ability to hide the sensor errors and abstract movements patterns into “tracks” may enable multiple applications. Indoor navigation (e.g., guiding a user to the milk shelf in a grocery store) is an example.

ConversationMaker

To be able to have a conversation with other people in the vicinity, can be useful. Two PhD students in a coffee shop, both pursuing their thesis on mobile computing, could engage into a conversation if they could learn about their mutual proximity. Similar situations occur in a gym, when visiting a friend’s wedding party, airports, etc. We imagine a Rainier application where people register with their profiles and interests, and the application provides a “matchmaking” service in real time. The matchmaking may be performed with an awareness of the context (a person may be more willing to talk about movies when she is in the movie theater). The privacy of individual can be preserved during the matchmaking process, and only revealed when both
parties are willing to meet. The *PathFinder* application mentioned above can be finally used to ensure that both people meet each other.

**VideoHighlights**

We envision a social application where mobile phones collaboratively sense their ambience and recognize “socially interesting” events. An outburst of laughter could be an acoustic trigger. Many people turning towards the wedding speech – detected from the correlated compass orientations of nearby phones – can be another example. Among phones that detect a trigger, the one with the “best quality” view of the event is shortlisted. At the end of the party, the individual recordings from different phones are correlated over time, and “stitched” into a single video highlights of the occasion. If done well, such a system could reduce the burden of manually editing a full-length video. Moreover, some events are often unrecorded in a social occasion, perhaps because no one remembered to take a video, or the designated videographer was not present at that instant. Such an app could be a solution. Rainier will provide a library of tools to aid the programmer in developing such an app.

Table 3 enlists a larger number of apps that may be easily developed in the Rainier environment. A number of these apps (marked with stars) have already been envisioned or prototyped in the research community.

### Table 1: Apps envisioned on the Rainier platform

<table>
<thead>
<tr>
<th>App</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WhatsOn</td>
<td>Events and activities in the vicinity appear on the phone in real time</td>
</tr>
<tr>
<td>Orchestra</td>
<td>People play collaborative music using mobile phone gestures</td>
</tr>
<tr>
<td>BusTimes*</td>
<td>Learning bus locations in real time</td>
</tr>
<tr>
<td>PotholePatrol*</td>
<td>Finding driving routes that are less bumpy</td>
</tr>
<tr>
<td>TrafficRoute*</td>
<td>Sensing traffic on roads</td>
</tr>
<tr>
<td>MobiUS*</td>
<td>Combining multiple phone displays to make a bigger display</td>
</tr>
<tr>
<td>ParkNet*</td>
<td>Finding parking spots quick</td>
</tr>
<tr>
<td>PointNConnect*</td>
<td>Sharing files by pointing</td>
</tr>
<tr>
<td>PEIR*</td>
<td>Stitching environment reports from mobile phones</td>
</tr>
<tr>
<td>CrowdCam</td>
<td>Seeing the world through other people’s cameras</td>
</tr>
<tr>
<td>VirtualBoard</td>
<td>Multiple people gesture with phones to paint on a virtual white board.</td>
</tr>
<tr>
<td>SignalMaps</td>
<td>Finding spots with good wireless (WiFi, GSM) connectivity</td>
</tr>
<tr>
<td>SoundPollute</td>
<td>Finding areas with low sound pollution</td>
</tr>
<tr>
<td>CrowdAsk</td>
<td>Ask around for answers</td>
</tr>
<tr>
<td>SocialGPS</td>
<td>Share our GPS to localize ourselves</td>
</tr>
</tbody>
</table>

Rainier modules execute sophisticated algorithms and data-intensive tasks, and reads and writes into the state blob as required. We describe the modules below.

#### 4.1 Virtual Sensors

Rainier provides a library of *Virtual Sensors*. A virtual sensor is a logical aggregation of several sensors and actuators, that are glued with software logic. An example is an “indoor/outdoor sensor” that is formed through the combination of actual light sensors, WiFi signal strength sensors, and suitable algorithms. Even though such an indoor/outdoor sensor does not exist in reality, such forms of abstractions is expected to enhance ease of programming. Other virtual sensors (VSs) can be created by “wrapping” existing VSs with additional sensors and software logic. Rainier will provide a library of such (hierarchical) VSs. Importantly, an expert in a specific VS can include her code in Rainier, facilitating other programmers to reuse them in new applications.

As a case study, we consider an *inPocket* virtual sensor, which determines whether a phone is inside the pocket. The pseudo code is presented below. The VS first accesses the time and location of the phone, and

---

1In this case, these services will be made available by Microsoft’s cloud, through a separate initiative called Hawaii. The advantage is that the traffic to and from these services will be confined to Microsoft’s data center, reducing the charges paid to ISPs. This cost reduction will benefit application designers.
Figure 2: The Rainier architecture: applications leverage the virtual sensor library while also accessing web services provided through a separate cloud service store (Microsoft’s Hawaii). The VS library in turn interfaces with managed data for the purposes of training the algorithms, calibration, etc. Also, Rainier communicates to mobile devices in the backend, and assimilates sensor measurements to compute the desired results. The Decision Engine optimizes device access, based on network connectivity, residual energy, and other (application-specified) utility functions.

then consults a whether service to determine whether that time is before or after sunset at that location. The phone’s light reading is then measured, and compared against different thresholds, depending on whether its day or night. Thresholds may be defined statically, however, to achieve higher accuracy, they may be trained based on light intensities in that location (more discussions later). Depending on the comparison, a true/false result is returned, along with a confidence value. The confidence computation is non-trivial because the physical errors in sensing needs to translate to the semantics of whether the phone is inside or outside the pocket. Put differently, it is not clear how 10% sensing error in the light intensity translates to a confidence on a binary variable. This is a clear research challenge with Rainier.

An indoor/outdoor virtual sensor may now be designed as a wrapper on the inPocket VS. The core logic is that when the phone is outside the pocket, the ambient light levels is a very reliable way to determine whether the user is indoors or outdoors. However, when the phone is inside the pocket, one has to rely on the quality of GPS readings to estimate the user location. The pseudocode for in/outdoor VS is also presented as a pseudo code, demonstrating how the using the inPocket sensor (from the VS library) makes programing easier.

**Algorithm 1** inPocket (phone i).

1: Loc = Location (i)  
2: getTime (Loc)  
3: Sunset = isAfterSunset (Loc, Time)  
4: Light = getLightMeter(i)  
5: if (Sunset == False)  
6: if (Light < Threshold1)  
7: return [True, Confidence]  
8: else return [False, Confidence]  
9: else // night time  
10: if (Light < Threshold2)  
11: return [True, Confidence]  
12: else return [False, Confidence]  

Emerging apps in location-based advertising is driving new localization technologies. For instance, SurroundSense [1] has shown the feasibility to recognize the logical location of a user (e.g., Starbucks, Wal-mart, airport), as opposed to its physical location (i.e., GPS
Algorithm 2 isIndoor (phone i).
1: if (inPocket (i) == False)
2:   Light = getLightMeter(i)
3:   if (Light > \tau_1 || Light < \tau_2)
4:     return [True, Confidence]
5: else return [False, Confidence]
6: else // phone is inside pocket
7:   G = getGPSLocation(i)
8:   if (G == NULL)
9:     return [True, Confidence]
10: else return [False, Confidence]

coordinate). However, SurroundSense has to be triggered only when users are within indoor environments. Other forms of localization technologies also require such triggers (otherwise, running indoor localization algorithms continuously can be prohibitive in terms of energy). We believe that an indoor/outdoor sensor would be a valuable primitive towards developing various forms of location technologies, a key focus area for future mobile computing applications.

Table 2 enlists an additional list of virtual sensors that may be supported as a part of the Rainier SDK. We anticipate a growth in such a sensor library as more application developers participate and contribute to it.

<table>
<thead>
<tr>
<th>Virtual Sensors</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car Driver</td>
<td>Is the user currently driving a car?</td>
</tr>
<tr>
<td>Alone</td>
<td>Is the user alone or in a group?</td>
</tr>
<tr>
<td>Queue</td>
<td>Queue length at a given place?</td>
</tr>
<tr>
<td>Happy</td>
<td>Is the atmosphere happy?</td>
</tr>
<tr>
<td>Watch TV</td>
<td>Is the user watching TV?</td>
</tr>
<tr>
<td>Activity</td>
<td>Is the user walking, running, dancing?</td>
</tr>
<tr>
<td>Shopping</td>
<td>Is the user shopping?</td>
</tr>
<tr>
<td>Busy</td>
<td>Is the user busy or open to suggestions?</td>
</tr>
</tbody>
</table>

Table 2: Examples of virtual sensors.

4.2 Managed Data

We observe that several apps need to bootstrap themselves with field data. For instance, WiFi based localization with Skyhook or Place Lab [2] requires the map of WiFi access points at different locations\(^2\). Acoustic data from humans may be necessary to train a gender classification app; accelerometer data from roads may be necessary to train a traffic congestion app. Single or small teams of app developers may not have the requisite resources to undertake such data gathering efforts, especially at city or country-wide scale. The typical approach, therefore, is to sense and record data at small scales, specifically tailored to the specific application’s requirement. Rainier aims to overcome this limitation through community formations with respect to sensed data. We observe the possibility of assimilating these islands of data, processing them as a whole, and creating a larger and richer data set. The idea is that the “whole is greater than the sum of the parts” – that is, the combined data set may better bootstrap each of the apps, while also enabling other apps that would otherwise be infeasible due to inadequate data. Figure ?? illustrates the idea with a diagram. The merged data set is distilled from two islands of data sets, and enables a third virtual sensor. Rainier will provide this data service, and interface it transparently with the virtual sensors. Addition of new data will continue to autonomously improve the system, with almost zero programmer involvement.

4.3 Sensor Abstraction Layer (SAL)

In an attempt to “decouple” computation from the phone platform (and only allow sensing to be platform specific), Rainier includes a Sensor Abstraction Layer (SAL). The SAL is instantiated in the cloud as well as on mobile phone, referred to as Upper SAL (USAL) and Lower SAL (LSAL), respectively. When the cloud-resident virtual sensors access physical sensors – such as getLightMeter() – the USAL converts these calls to remote procedure calls (RPCs) and exports them to the phones. The LSAL interprets these calls, translates them into platform specific calls to sensors, and gathers the readings. For instance, for windows mobile phones on Samsung vendors, the precise function call to the light sensor may be different from that of OS4 on iPhones. The LSAL executes this call, and forwards the results to a module called Operators (described later). The Rainier programmer remains agnostic of the (heterogeneous) sensing platform – her task simplifies to making a high level call, e.g., getLightMeter() or getCompassReading() .
The idea behind separating computation from sensing stems from the potential advantages of using a unified cloud environment for all computation-bound tasks. A natural question is why not use Javascripts that can execute on the phone platform. Of course, Javascripts are certainly an option, however, current versions may not support access to sensors on mobile devices. While future programming support is expected to include such supports (e.g., HTML 5), we believe that several virtual sensors and application logic may be too heavyweight to run client-side. As a result, Rainier will design its framework such that virtual sensors (or parts of them) are intelligently partitioned between the cloud and the phone. Some VSs will reside on the phone if it appears that executing that sensor on the phone is always more appropriate. The partition will be driven both by the complexity of the algorithm as well as the sensing data that must be transferred to complete that operation.

4.4 Operators
Exporting raw sensor measurements to the VSs in the cloud can be energy and bandwidth inefficient – Operators are used to summarize the data. To this end, operators are statistical functions that summarize the sensor readings in a manner that the preserves the correctness of the VS’s operation. Thus, an operator could perform simple functions like addition, average, maximum, variance, etc. To be specific, operators can also be viewed as a function pointer to a VS on the phone, which performs more complicated operations. Thus, the LSAL forwards the sensing results to the operators; the output of the operators are returned to the USAL.

4.5 Decision Engine
Finally, Rainier also provides a Decision Engine (DE) that is responsible for optimizing system operations. For example, if an application or a virtual sensor intends to sample 50% of the phones in a specific region, the DE could attempt to sample phones that are already active. The DE may even account for residual energy of each phone, network connectivity, or other utility functions, in resolving the query. In performing these operations, the DE interacts with the virtual sensors, the application, as well as the USAL.

4.6 Case Study
As a case study, we take up an app that computes the number of people in a specified building. This may be useful when a person intends to consider whether many others have gone to office on a snow day or a weekend; this may also be useful in launching product advertisements in a mall. The pseudocode for this app is presented below. We contend that given the indoor/outdoor virtual sensor, the app becomes easy to write with a few lines of code. In this case, the app first computes the GPS location of the building, and then consults a IT database to obtain all the WiFi access points in that building. Then, the app queries a registration database that returns all the phones associated to any of these WiFi APs. For each of these phones, the app queries whether it is located indoor or outdoors; the count of indoor phones is returned.

**Algorithm 3** HowManyInside ((loc = Building 99))

```
1: G = getGPS (loc)
2: W[...] = getWiFi-AP-Database (G)
3: S[...] = phonesAssociated2WiFi (W)
4: for phone i in S[...]
5:     test = isIndoor (i)
6:     if (test == TRUE)
7:         total++
8: return total
```

Emerging apps are beginning to access an increasing number of web services. For instance, a simple restaurant search app accesses Skyhook databases for WiFi maps, Google maps for visualization, and perhaps yellow pages to superimpose restaurants in that area. Access to such services incur excessive signaling costs that are undesirable to wireless service providers (who are already facing severe spectrum pressure). Further, the mobile device also expends energy in pulling information from all these services. We observe that Rainier alleviates both signaling and energy by consolidating all these operations on the cloud. Figure 4 shows the timeline of operations for the HowManyInside app. The accesses to getGPS, getWiFi-AP-Database, phonesAssociated2WiFi all traverse over the wired network, and in the case of a single-vendor cloud, remains within the same data-center. The only wireless traffic pertains to the access of physical sensors from all the phones; however, this traffic would have been present even if the app was running on the phones and reporting indoor/outdoor readings to the cloud for aggregation. In sum, Rainier reduces the signaling and energy costs for a class of applications that rely on collaboration among devices and multiple accesses to web services. We believe this aligns with the interest of wireless service providers.

5. DISCUSSION
Translating these ideas into a scalable, deployable system entails a wide variety of inter-disciplinary research challenges. For instance, (1) designing a catalog of virtual sensors requires creative ideation and mashing of sensors and algorithms. (2) When designing any VS, the uncertainty in physically sensed data needs to be interpreted semantically (e.g., how does noise in an accelerometer reading translate to error in the “car
driver” sensor). (3) How can data collected by different applications be cleaned and merged effectively. (4) In the interest of scalability, how can Rainier be made entirely stateless. (5) What are the appropriate metrics for evaluating the success of Rainier. These and many other questions underlie the Rainier architecture.

6. CONCLUSION

The mobile computing industry is experiencing a rising popularity in people-centric sensing applications. Today’s applications are mostly simple and run independently on the phones. We envision that the next generation of applications will be collaborative and resource-rich – apps would run over scattered devices and demand access to powerful CPUs, data, and sophisticated algorithms. Rainier is a programming environment that is designed to meet these needs. By assimilating a library of virtual sensors – a form of sensor abstraction – programmers can perform complex tasks through simple function calls. Sensed data from multiple app developers are also assimilated and presented in a “managed” form, facilitating operations such as system calibration, training, and learning. Finally, programming in the cloud empowers the app developer to harness Microsoft’s rich software development tools, while providing a bird’s eye view of the scattered sensors. Thus, in this new programming model, mobile phones are truly the backend of computation.

7. REFERENCES