# Programming Support for Sharing Resources Across Heterogeneous Mobile Devices

Zheng Song, Sanchit Chadha, Antuan Byalik, and Eli Tilevich
Software Innovations Lab, Virginia Tech
{songz,schadha,antuanb,tilevich}@cs.vt.edu

## ABSTRACT

Modern mobile users commonly use multiple heterogeneous mobile devices, including smartphones, tablets, and wearables. Enabling these devices to seamlessly share their computational, network, and sensing resources has great potential benefit. Sharing resources across collocated mobile devices creates mobile device clouds (MDCs), commonly used to optimize application performance and to enable novel applications. However, enabling heterogeneous mobile devices to share their resources presents a number of difficulties, including the need to coordinate and steer the execution of devices with dissimilar network interfaces, application programming models, and system architectures. In this paper, we describe a solution that systematically empowers heterogeneous mobile devices to seamlessly, reliably, and efficiently share their resources. We present a programming model and runtime support for heterogeneous mobile device-to-device resource sharing. Our solution comprises a declarative domain-specific language for device-to-device cooperation, supported by a powerful runtime infrastructure. we evaluated our solution by conducting a controlled user study and running performance/energy efficiency benchmarks. The evaluation results indicate that our solution can become a practical tool for enhancing the capabilities of modern mobile applications by leveraging the resources of nearby mobile devices.

**ACM Reference Format:**
Zheng Song, Sanchit Chadha, Antuan Byalik, and Eli Tilevich. 2018. Programming Support for Sharing Resources Across Heterogeneous Mobile Devices. In *Proceedings of MOBILESoft '18: 5th IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft '18)*. ACM, New York, NY, USA, Article 4, 12 pages. https://doi.org/10.1145/3197231.3197250

## 1 INTRODUCTION

The modern computing landscape is marked by several rapidly evolving realities. A typical user owns multiple mobile devices that differ in their types, platforms, and capabilities. For example, a user may simultaneously own a smartphone, a tablet, an e-reader, each of which runs a different operating system and offers vastly dissimilar processing capabilities, sensory functionalities, and networking interfaces. Furthermore, the number and variety of mobile devices in a typical household is even greater. Finally, the rapid developments in wearable computing and the Internet of Things (IoT) have the potential to increase these numbers for a typical user by as much as an order of magnitude in the near future.

Mobile devices have traditionally used the cloud as a means of enhancing their execution [25, 26, 44], both to improve the quality of service and to extend their functionality. Nevertheless, accessing cloud-based resources is not always feasible, beneficial, or safe. On the other hand, with the rapid growth of capacity of mobile devices, the computational power could be provided by nearby mobile devices instead. All these scenarios give rise to the potential of leveraging nearby mobile devices, often owned by the same user or a community of users, as an alternative means of gaining additional resources.

### 1.1 Motivating Scenarios

Figures 1, 2 and 3 depict three scenarios exemplifying the conditions described above. In Figure 1, a smartphone application needs to search for a given face from all photos in the phone's album. Facial recognition is known to be computation/energy-intensive thus causing high latency/battery consumption, especially when the user has hundreds of photos. In Figure 2, the driver is navigated through a smart glasses interface. However, keeping the glasses' GPS module on continuously could drain the device's battery quickly. In Figure

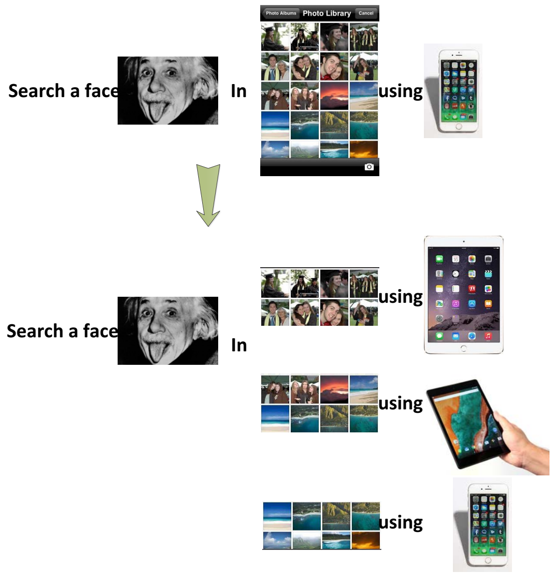


Figure 1: Scenario 1: Photo Recognition

*MOBILESoft '18, May 27–28, 2018, Gothenburg, Sweden*
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-5712-8/18/05...$15.00
https://doi.org/10.1145/3197231.3197250

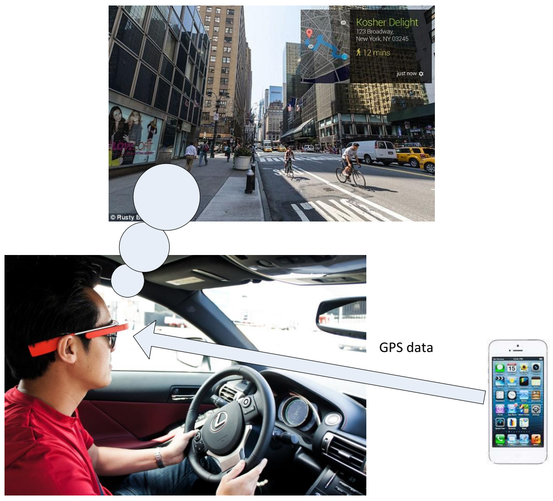


**Figure 2: Scenario 2: GPS Sharing**

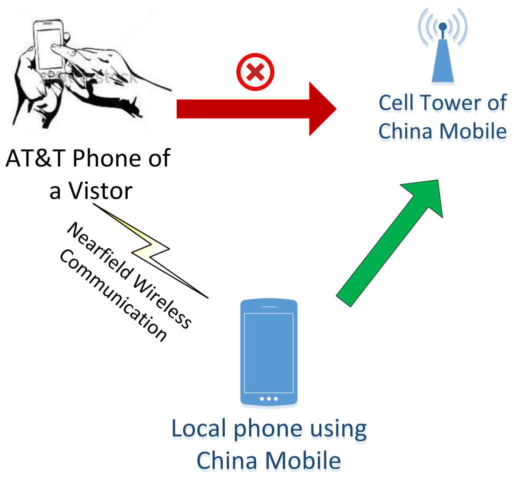

AT&T Phone of a Vistor
Cell Tower of China Mobile
Nearfield Wireless Communication
Local phone using China Mobile

**Figure 3: Scenario 3: Data Plan Sharing**

3, a smartphone user on a short-term trip to a foreign country needs to access the Internet. However, without a local mobile account, the phone cannot access any mobile data services provided by the available cellular network providers.

Although the users mentioned above are all short of either computational resources, context-related resources, or network resources, various mobile devices (e.g., tablets, ereaders, and wearables, etc.), owned by themselves or their acquaintances may be in the immediate vicinity. These devices could provide the external resources required to solve the problems above. One could rewrite the mobile applications, so as to enable them to take advantage of such external resources. In scenario 1, one can reduce the execution time of computationally intensive tasks if they are run in a piecemeal fashion on nearby devices. In scenario 2, one can request GPS sensory reading from a nearby mobile device with larger battery capacity. In scenario 3, one can access the Internet by using a nearby mobile device, with a local mobile data plan, as a proxy that forwards the network requests and responses.

## 1.2 Research Challenges and Contribution

The aforementioned scenarios demonstrate how by sharing the resources of nearby devices, mobile applications can not only improve their quality of service, but also provide new functionality. However, several conceptual obstacles stand in the way of such resource sharing across heterogeneous mobile devices. For example, in scenario 1, one cannot execute offloaded mobile functionality on a different platform (e.g., running Android code on iOS). In scenario 2, one needs to be able to dynamically locate a nearby mobile device, whose battery capacity can accommodate long-lasting GPS sensor reading. In scenario 3, the programming interface to another user's mobile device must provide access to the device's voluntarily shared resources, while preventing misuse. The runtime in all scenarios must properly adapt to the mobility of the devices involved, ensuring efficiency and robustness.

The prior state of the art has studied novel applications of sharing resources across nearby mobile devices. However, these solutions mostly have focused on specific mobile platforms, without the overarching goal of supporting heterogeneous environments. These prior solutions have lacked focus on programmability and thus require the programmer to write complex logic for error handling and performance/energy consumption optimization.

In this paper, we present solutions that address the deep, conceptual challenges of enabling mobile devices to provide/use resources for/of nearby heterogeneous mobile devices. These solutions embrace heterogeneity, working with any pair of mobile devices, irrespective of their platforms, operating systems, or installed applications. Also, the presented solutions reduce the programmer's effort in creating reliable and efficient functionality for sharing resources. This paper makes the following contributions:

- We study and reveal how existing applications can benefit from shared resources of nearby devices.
- We design the Resource Query Language (RQL)—a declarative domain-specific language for accessing shared resources of nearby devices. RQL makes it possible to declaratively express resource sharing requests by simply specifying the preferred devices, resource types, and the actions to be carried out. The RQL runtime is designed with provisions for energy efficiency, latency optimization, and privacy preservation when executing across heterogeneous mobile devices.
- We provide a reference implementation of the RQL language and runtime support on major mobile platforms, including iOS and Android. We also describe example applications that make use of RQL to access resources across the iOS and Android platforms
- We evaluate the programmability and efficiency of our technical approach through a case study and experiments. Our results indicate that the presented solutions can improve the productivity of mobile programmers, as well as improve the performance/energy efficiency of mobile applications.

The rest of this paper is organized as follows. Section 2 studies the functionality of existing applications that can benefit from resource sharing. Section 3 introduces our design of the resource sharing solution, focusing on the proposed RESTful language for the programmers to specify their resource requirements. Section 4 describes the design of the runtime support, and the optimization strategies we designed for energy efficiency, latency optimality, and privacy preservation. Section 5 introduces our reference implementation of RQL and its runtime. We also describe how we have applied RQL to realize the solutions motivated in this section. Section 6 shows how our resource sharing method benefits the programmers, as well as improve the performance/energy efficiency of mobile applications. Section 7 summarizes the related state of art, and Section 8 concludes this paper and puts forward some future research directions based on this work.

## 2 IDENTIFYING REQUIREMENTS

In this section, we demonstrate the potential benefits of resource sharing across mobile devices by answering these two questions: 1) for existing apps, how many kinds of local API calls may possibly be replaced with remote calls? 2) how frequently such replaceable APIs are called in existing applications?

### 2.1 Methodology

In the study, we use 574 of most popular Android applications in different application domains, which we downloaded from Google Play in September 2014. We carefully analyzed the APIs included in these applications, and found that those APIs which can benefit from nearby resources can be classified into **three major categories: (1) context-providing sensors/media tools (e.g., GPS, accelerometer, microphone, camera); (2) computational resources (e.g., processors, memory, and storage); (3) service resources (e.g., network connections, phone service, SMS).**

Following the steps given below, we pick the API calls that belong to the above listed three categories.

1) We disassemble the deployment archives of these applications, and use a tool called Baksmali to de-compile Android DEX (VM bytecode) files into Smali files (readable code in the smali language) which can be analyzed. Regular expressions are then used to pick out all API calls, as shown in Fig 4. (a).

2) We then remove the APIs of the packages irrelevant for resource sharing, such as java/lang/, java/io/, com/google/ads/, android/view; android/os/. Fig 4.(b) shows the remaining APIs.

3) Finally we manually remove those APIs that do not fall into the three considered categories above. Fig 4. (c) shows the left APIs, in which those marked in black are the APIs that involve service resources, those marked in green stand for APIs that need computational resources, and those marked in yellow show APIs that are sensor related.

### 2.2 Results

We randomly picked applications from 3 application domains (Book, Business, and Game) and listed their API usages in Table. I. From the table we can see: 1) The APIs that provide HTTP services, including *webview⟶loadurl*, *url⟶openconnection*, *httpurlconnection⟶connect*, *httpclient⟶execute*, are widely used in every application domain. By cooperatively providing HTTP services for a group of devices, one can reduce the total energy consumption of the group because some contents can be shared among nearby devices[18]. Consider cellular links, which are energy intensive at low bit-rates and have high round-trip times after idle periods. Here consolidating multiple users' traffic on a subset of links would shorten the round trip time as well as save the energy consumption[34]; 2) The APIs for local database searching and crypto are frequently spotted in game applications. Such APIs consume more computational power than other APIs, so if they can be executed by another device with greater computational power, it will save the overall energy consumption and speed up the whole execution[1]; 3) The APIs for obtaining sensors' readings are frequently spotted in book and game applications. As commercial sensors usually will not provide enough accuracy to figure out the attitude of the phone and the exact motion of users[48]; using other sensors from a nearby device can provide a viable alternative.

**Table 1: Offloadable APIs**

| | Book | Business | Game |
|---|---|---|---|
| Number of applications | 27 | 27 | 18 |
| Services per application | 2.7 | 2.7 | 3.6 |
| Computational-intensive: database | 0.59 | 0.48 | 0.5 |
| Computational-intensive: crypto | 0.59 | 0.37 | 0.83 |
| Context-related API: sensor | 0.29 | 0 | 0.67 |
| Specific API 1: gestures recognition | 0.41 | 0 | 0 |
| Specific API 2: sslconnection | 0 | 0.26 | 0 |
| Specific API 3: khronos (OpenGL) | 0 | 0 | 0.56 |

By carefully analyzing existing applications, one can conclude that most applications in different domains can generally benefit from using the network service/ computational / sensory resources of nearby devices. In the following sections, we will detail our solutions that enable such resource sharing across nearby devices.

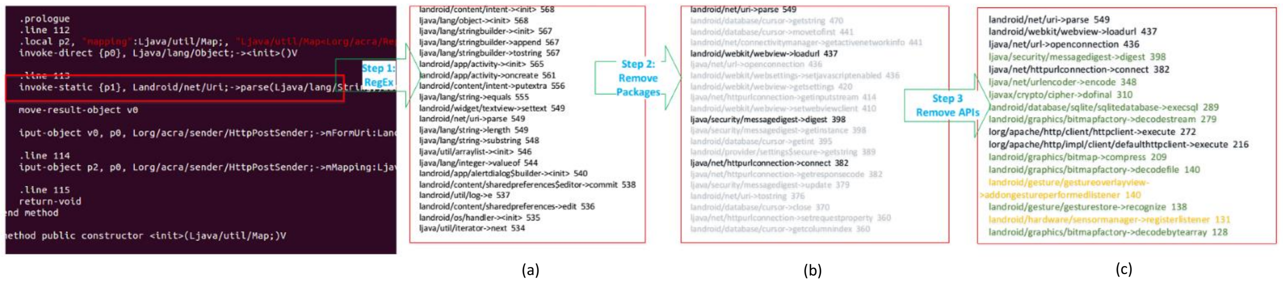
## 3 RQL DESIGN

In this section, we present the design of the resource sharing language (RQL). RQL is a platform-independent, domain-specific language that enables heterogeneous devices to seamlessly share their resources. We designed RQL around the RESTful architecture [12], a proven solution for many of the complexities of engineering dynamic, heterogeneous distributed systems, including the WWW.

In our target domain, we leverage the flexibility of this architecture to hide the complexity of the inherent heterogeneity of mobile devices that need to participate in device-to-device resource sharing scenarios. We observe that in this domain, the actual operations on the shared resources are limited to a small set, and exploit this observation to provide a concise yet powerful DSL for resource sharing. Specifically, the design of RQL follows the verb/nouns paradigm: nouns express the requested resources, while verbs express the actions performed on these resources.

We next present RQL by example. Consider an RQL statement: `pull glass:sensor/orientation`. This statement will retrieve the readings of the orientation sensor of a glass device, if it happens to be in the vicinity; it will return a `null` reading otherwise. The specific details of locating a glass device, connecting to it, retrieving its readings, etc. are handled by the RQL runtime.

*3.0.1 Nouns.* RQL represents the resource intention with nouns. Specifically, the nouns comprise the following parts: "device description:resource description/specific name".



```
    .prologue
    .line 112
    .local p2, "mapping":Ljava/util/Map;, "Ljava/util/Map<Lorg/acra/Re
    invoke-direct {p0}, Ljava/lang/Object;-><init>()V

    .line 113
    invoke-static {p1}, Landroid/net/Uri;->parse(Ljava/lang/Strin

    move-result-object v0

    iput-object v0, p0, Lorg/acra/sender/HttpPostSender;->mFormUri:Lan

    .line 114
    iput-object p2, p0, Lorg/acra/sender/HttpPostSender;->mMapping:Ljav

    .line 115
    return-void
nd method

ethod public constructor <init>(Ljava/util/Map;)V
```

```
landroid/content/intent-><init> 568
ljava/lang/object-><init> 568
ljava/lang/stringbuilder-><init> 567
ljava/lang/stringbuilder->append 567
ljava/lang/stringbuilder->tostring 567
landroid/app/activity-><init> 565
landroid/app/activity->oncreate 561
landroid/content/intent->putextra 556
ljava/lang/string->equals 555
landroid/widget/textview->settext 549
landroid/net/uri->parse 549
ljava/lang/string->length 549
ljava/lang/string->substring 548
ljava/util/arraylist-><init> 546
ljava/lang/integer->valueof 544
landroid/app/alertdialog$builder-><init> 540
landroid/content/sharedpreferences$editor->commit 538
landroid/util/log->e 537
landroid/content/sharedpreferences->edit 536
landroid/os/handler-><init> 535
ljava/util/iterator->next 534
```

```
landroid/net/uri->parse 549
landroid/database/cursor->getstring 470
landroid/database/cursor->movetofirst 441
landroid/net/connectivitymanager->getactivenetworkinfo 441
landroid/webkit/webview->loadurl 437
ljava/net/url->openconnection 436
landroid/webkit/websettings->setjavascriptenabled 436
landroid/webkit/webview->getsettings 420
ljava/net/httpurlconnection->getinputstream 414
landroid/webkit/webview->setwebviewclient 410
ljava/security/messagedigest->digest 398
ljava/security/messagedigest->getinstance 398
landroid/database/cursor->getint 395
landroid/provider/settings$secure->getstring 389
ljava/net/httpurlconnection->connect 382
ljava/net/httpurlconnection->getresponsecode 382
ljava/security/messagedigest->update 379
landroid/net/uri->tostring 376
landroid/database/cursor->close 370
ljava/net/httpurlconnection->setrequestproperty 360
landroid/database/cursor->getcolumnindex 360
```

```
landroid/net/uri->parse 549
landroid/webkit/webview->loadurl 437
ljava/net/url->openconnection 436
ljava/security/messagedigest->digest 398
ljava/net/httpurlconnection->connect 382
ljava/net/urlencoder->encode 348
ljavax/crypto/cipher->dofinal 310
landroid/database/sqlite/sqlitedatabase->execsql 289
landroid/graphics/bitmapfactory->decodestream 279
lorg/apache/http/client/httpclient->execute 272
lorg/apache/http/impl/client/defaulthttpclient->execute 216
landroid/graphics/bitmap->compress 209
landroid/graphics/bitmapfactory->decodefile 140
landroid/gesture/gestureoverlayview-
>addongestureperformedlistener 140
landroid/gesture/gesturestore->recognize 138
landroid/hardware/sensormanager->registerlistener 131
landroid/graphics/bitmapfactory->decodebytearray 128
```

**Figure 4: Steps of Picking APIs**

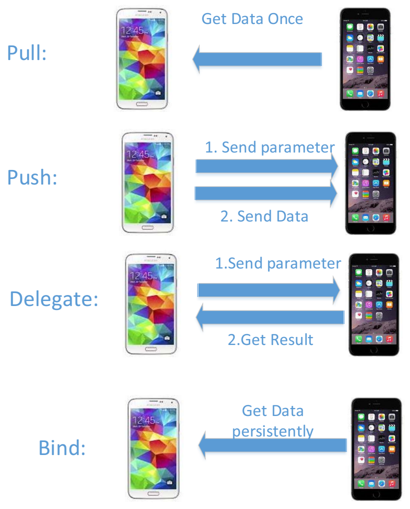


**Figure 5: Defined RQL Verbs**

Device description defines device types (e.g., glass, smartphone, tablet, etc.) or specific characteristics (e.g., name, owner, OS, etc.). Resource description defines the type of resource (e.g., sensors, services, files, etc.) followed by specific names (e.g., sensor/orientation, sensor/gps, service/facerecognition, service/httpsend, etc.).

*3.0.2 Verbs.* Following the RESTful design principles, a small number of verbs manipulates an infinite number of nouns. In particular, RQL defines only four verbs: `pull`, `push`, `delegate`, and `bind`. As shown in Fig. 5, "`pull`" retrieves data from the service interface of another device immediately; "`push`" sends data from the source device to the target device; "`delegate`" sends some parameters and then gets the execution results back; finally, "`bind`" establishes a persistent connection to a device to obtain the value changes of a specific sensor.

*3.0.3 Adverbs.* Although traditional RESTful interfaces consists of only verbs and nouns, RQL integrates adverbs as informed by some prior research on fault-tolerant RESTful services [11]. In RQL, adverbs can express how commands should be executed in terms of time or quality constraints. For example, an adverb can express the timeout value for a `pull` command (in ms) (e.g., `pull external:alg/OCR -latency < 500ms`). Another adverb is `-blocking` (e.g., `pull external:sensor/GPS -blocking`, which expresses that the RQL call to retrieve the GPS reading should block, to return only when a GPS reading becomes available or the call has failed. By default, all RQL statements are non-blocking with the results communicated via an asynchronous callback mechanism. We discuss a programming scenario involving the `-blocking` adverb in Section 5.

Fig. 6 depicts several examples of using RQL. The first example is concerned with getting GPS readings from another device. The second example sends a data file to a remote device (belonging to user John) to use as a parameter to a facial recognition algorithm. The third example directs a remote device to perform an HTTP request for a given URL and sends back the obtained output. The fourth example establishes a persistent connection to get orientation sensor updates from John's smart glasses device.

Sometimes the source device may need to execute a sequence of RQL statements on the same target device consecutively. To that end, RQL features the "|" binary operator, which specifies that its operands are to be transmitted in bulk to the target device and executed in sequence. Consider the source device needing to execute both the OCR and language translation algorithms one after another on the same target device. The programmer can express this functionality in RQL as shown in line 2 of Fig. 6. Batching RQL requests may also reduce their aggregate latency.

## 4 RUNTIME DESIGN

To meet its design goals, RQL requires sophisticated runtime support for mainstream platforms (i.e., Android, iOS, and Windows Phone). In this section, we identify the requirements and outline design of such runtime support. With respect to requirements, the RQL runtime must reconcile the need for efficiency with that of portability and ease of implementation. Hence, we have deliberately constrained our runtime design to the application space, so as to avoid low-level, platform-specific system changes. In other words, the user should be able to install the RQL runtime as if it were a regular mobile application, albeit with extended permissions (e.g., access to all sensors, the ability to connect to remote services via all available network interfaces, access to local application data and external storage, etc.)

```
1. pull any:sensor/GPS
2. push John:file/myphoto.jpg -i /DCIM/20170319323.jpg | delegate service/faceRecognition
3. delegate any:service/http -t http://www.google.com
4. bind John/glasses:sensor/orientation
```

**Figure 6: RQL Examples**

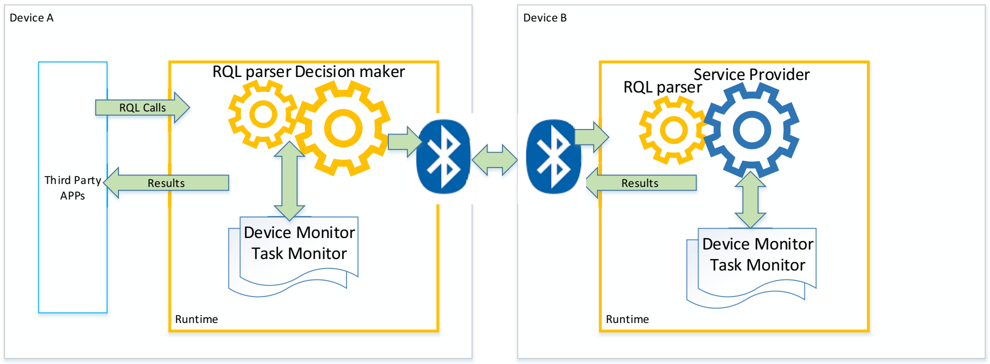


**Figure 7: General Design of Runtime Support**

The runtime support, whose basic flow appears in Figure 7, includes three basic modules: client, server, and monitor. The client module of Device A accepts an RQL request and determines whether the request can be executed by a nearby device (Device B) by querying a distributed registry of nearby devices and resources they provide. The devices communicate by means of near field communication interface (e.g., Bluetooth). The server module of Device B parses the request, executes it, and returns the result back to the client module of Device A. The monitor module comprises two parts: device and service status. The device status monitor keeps track of the battery levels, resource usage status, and locations of nearby devices. The service status module monitors the energy consumption/latency of the services provided by the nearby devices.

In the remainder of this section, we will further demonstrate how we optimized the runtime design for energy efficiency, latency reduction, and privacy preservation.

## 4.1 Ensuring Energy Efficiency

*4.1.1 Choosing Communication Channels.* In our runtime design, Bluetooth Low Energy (BTLE) serves as the major communication mechanism for two reasons: 1) BTLE is known to be the most energy efficient way to discover/announce external services. Although WiFi and Bluetooth are popular device-to-device communication mechanisms, their energy consumption levels are larger than that of BTLE, both in active and idle modes; 2) to support heterogeneity, the runtime must be able to use a communication mechanism supported by major mobile platforms. Mainstream mobile communication mechanisms, including WiFi-direct and traditional Bluetooth, cannot connect a recent (i.e., 4.4.2 and up) Android device with an iOS device.

However, BTLE does have some limitations. Chief among them is the primary use-case for BTLE: command transmission and small data-size transmissions. The main purpose of BTLE is to send small bursts of data for extended periods of time while consuming minimal energy. The largest size package BTLE will send is 20 bytes. Therefore, when the runtime needs to send a data file to another device, using a different communication mechanism can provide performance advantages.

To overcome the limitations of BTLE when transferring larger data volumes, our design includes an optimization that makes use of edge servers. When transferring a data file, the runtime at the source device uploads the file to an edge server, and send the URL of that file to the target device via a BTLE connection for the target device to download. Nevertheless, it is worth noting that, with both Android and iOS constantly improving the relatively new inter-device communication mechanisms, our runtime is capable of communicating via WiFi-direct, once it becomes available for heterogeneous devices.

*4.1.2 Choosing Target Device.* When multiple devices can be used for a given task, selecting the correct device could save the overall energy consumption of all devices. For tasks that require the service to send HTTP requests, as the 3G chips would still cost energy when the data transmission is finished, combining multiple requests and sending them at once could greatly save the overall energy consumption. For tasks that require a specific sensory reading like GPS, the major energy consumption happens when the target device tries to obtain the sensory reading. Therefore, combining multiple sensory requirement tasks to the same device could also reduce the overall energy consumption.

We intend to use an incentive strategy to encourage batching HTTP requests and sensory data requests to the same target device. The basic idea is to let the device which has already been the delegation of such requests to ask for lower bid prices for other tasks of the same kind. The details are described in Sec. 4.3.

### 4.2 Reducing Latency

Different from the HTTP requests and sensory data requests, RQL requests which need to perform computationally intensive tasks can not be energy-optimized by being batched to a same delegation. On the contrary, when such tasks are combined to the same target device, their time/latency usually gets larger. Therefore, in the runtime, for those RQL requests that want to process an amount of computation intensive tasks through multiple devices, the runtime needs to act as the load balancer: it needs to divide the necessary tasks into chunks between multiple devices in a way that the overall waiting time is minimized.

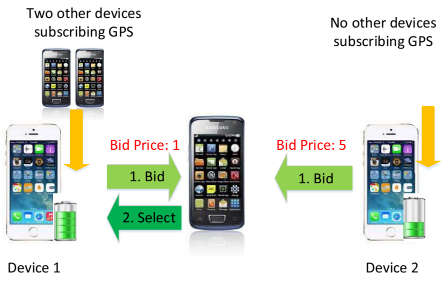
The most accurate way to balance loads across numerous available devices is to get real-time loads from each devices and also the execution time of each task in advance. However, the frequent communication among devices costs extra energy and clogs the channel as it is occupied for a larger amount of time. In such cases, the solution we take is to log the load of each device in the format of how many tasks are running or waiting. The running tasks of surrounding devices are updated through the device monitor's scanning action. When the runtime assigns one task to a device, the device's load of is incremented by one; when it get the result back from a device, its load is decremented by one. Therefore, each time when the runtime needs to assign a task, it assigns it to one of all the devices providing that service with the lowest load.

### 4.3 Incentive Strategy

In the presence of multiple unrelated mobile users, the adopters of this technology may face the problem of having to motivate them to share the resources of their devices. One possible approach is putting in place an incentive strategy that employs microtransactions for devices to pay for the external resources consumed. The payments can be represented as marketplace credits to pay for using shared resources in the future or even as a standard currency.

The RQL runtime's design includes an incentive strategy based on the reversed auction model, as shown in Fig. 8. Before a device can issue an RQL request, the runtime scans all nearby devices and gets their bid prices for the required service. It then chooses a device with the lowest bid price as the target of offloading. When other devices have the same bid prices, it randomly picks one, or chooses one according to their loads. After the task is finished and the results are returned, it pays the chosen device the bid price as incentive. This strategy would help motivate unrelated users to make the resources of their devices available for sharing.

An incentive strategy can also take energy consumption into consideration when offering bids. For example, a mobile device already delegating HTTP or sensory tasks, should be able to offer lower bid prices than idle devices, as performing additional tasks would incur smaller energy costs. Therefore, the probability of forwarding the majority of HTTP requests or sensory reading tasks to the same device would increase. In such cases, the energy consumption of all the participating devices becomes minimized. Hence, the initial investment into recruiting mobile users to participate in resource sharing will be amortized by the future improvements in usability and performance. Incentive strategies thus constitute a promising future research direction for this work.


Two other devices subscribing GPS
No other devices subscribing GPS
Bid Price: 1
1. Bid
2. Select
Bid Price: 5
1. Bid
Device 1
Device 2
Figure 8: Flow of Reversed Auction

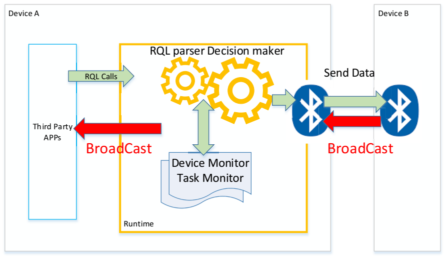


Figure 9: Possible Attacks

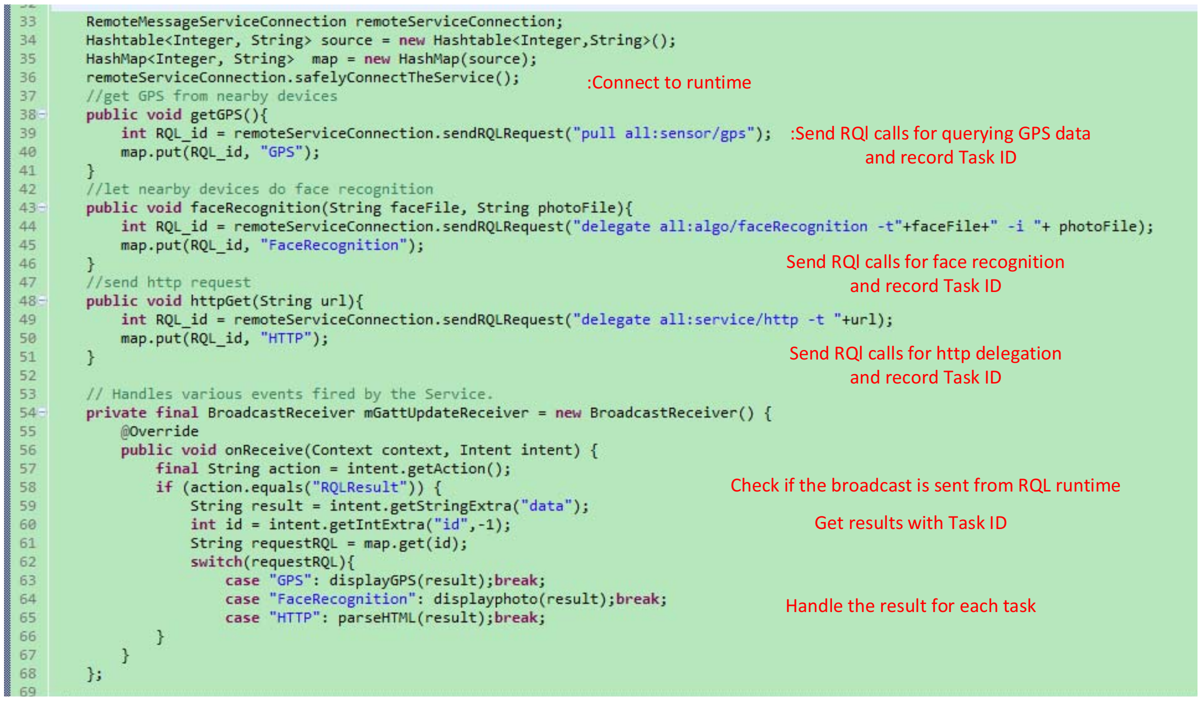
### 4.4 Privacy and Security

Fig. 9 describe the potential threat of privacy leakage and security issues, where device A and device B are the source and the target, respectively. The security threats could arise in the following scenarios: 1) When the runtime on device A broadcasts the result of some third party application, it could be wiretapped by a malware installed on that device. 2) when the runtime on device A receives the broadcast from device B through Bluetooth, another device C binding to the same Bluetooth channel might get that message as well. One can counter this security threat by encrypting the message. To solve the problem, the third party application will need to provide a public encryption key for each RQL request, so that the runtime can encrypt the result with that key. This way, it is only the third party application with the private key that can decrypt the result. Although our reference implementation does not yet include this security mechanism, our design makes it possible to straightforwardly add it to the runtime.

## 5 REFERENCE IMPLEMENTATION

Our reference implementation of RQL and its runtime concretely reifies the design decisions we described in Sections 3 and 4. While we have implemented all the described features of RQL including the required runtime support, some of the optimization and privacy provisioning features of the runtime remain a work in progress.

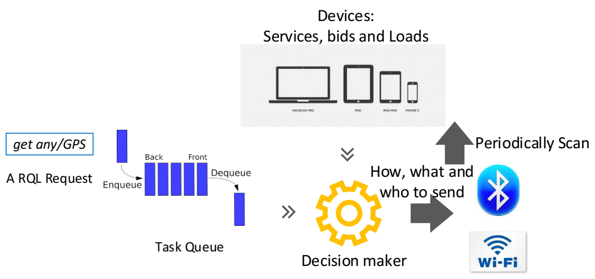
To demonstrate our implementation, we next describe how we used it to address the resource sharing needs in the three motivating examples from Section 1. The snippets of Java code in Fig 10 show how the three source devices use RQL to access resources of nearby target devices. Fig 11. (a) gives an overview of the communication flow between the source device's applications and the runtime, while Fig 11. (b) shows a sequence diagram of an iOS device communicating with an Android device by means of RQL.



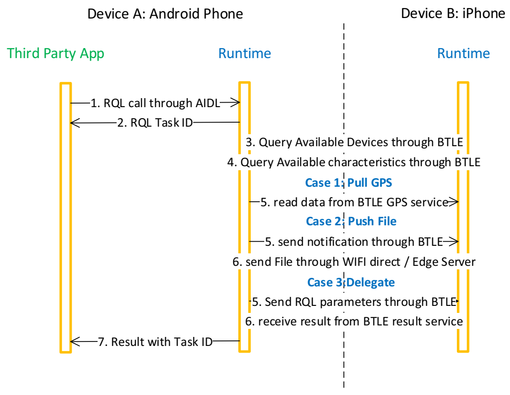
```
    RemoteMessageServiceConnection remoteServiceConnection;
    Hashtable<Integer, String> source = new Hashtable<Integer,String>();
    HashMap<Integer, String>  map = new HashMap(source);
    remoteServiceConnection.safelyConnectTheService();
    //get GPS from nearby devices
    public void getGPS(){
        int RQL_id = remoteServiceConnection.sendRQLRequest("pull all:sensor/gps");
        map.put(RQL_id, "GPS");
    }
    //let nearby devices do face recognition
    public void faceRecognition(String faceFile, String photoFile){
        int RQL_id = remoteServiceConnection.sendRQLRequest("delegate all:algo/faceRecognition -t"+faceFile+" -i "+ photoFile);
        map.put(RQL_id, "FaceRecognition");
    }
    //send http request
    public void httpGet(String url){
        int RQL_id = remoteServiceConnection.sendRQLRequest("delegate all:service/http -t "+url);
        map.put(RQL_id, "HTTP");
    }

    // Handles various events fired by the Service.
    private final BroadcastReceiver mGattUpdateReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            final String action = intent.getAction();
            if (action.equals("RQLResult")) {
                String result = intent.getStringExtra("data");
                int id = intent.getIntExtra("id",-1);
                String requestRQL = map.get(id);
                switch(requestRQL){
                    case "GPS": displayGPS(result);break;
                    case "FaceRecognition": displayphoto(result);break;
                    case "HTTP": parseHTML(result);break;
                }
            }
    };
```

:Connect to runtime
:Send RQl calls for querying GPS data and record Task ID
Send RQl calls for face recognition and record Task ID
Send RQl calls for http delegation and record Task ID
Check if the broadcast is sent from RQL runtime
Get results with Task ID
Handle the result for each task

**Figure 10: Mobile Application Code using RQL**



(a) Communication Flow


Device A: Android Phone
Device B: iPhone
Third Party App
Runtime
Runtime
1. RQL call through AIDL
2. RQL Task ID
3. Query Available Devices through BTLE
4. Query Available characteristics through BTLE
Case 1: Pull GPS
5. read data from BTLE GPS service
Case 2: Push File
5. send notification through BTLE
6. send File through WIFI direct / Edge Server
Case 3: Delegate
5. Send RQL parameters through BTLE
6. receive result from BTLE result service
7. Result with Task ID

(b) Runtime Sequence Diagram

**Figure 11: Third Party Application**

On Android, the RQL runtime executes as a background service. Android applications communicate with the runtime by establishing an Android Interface Definition Language (AIDL) connection, a standard Android mechanism for inter-application/service communication. The Android API provides methods for sending RQL requests over the established AIDL connection. Upon receiving an RQL request, the runtime immediately returns a unique identifier for that request. The application can then use this identifier to locate the request's results once it has been carried out. The runtime is responsible for several functionalities, including parsing the RQL commands, determining which device should be the target for a given command, controlling the communication (over Bluetooth LE) with other devices, and receiving the results from target devices. The returned results are made available to mobile applications via a broadcast-based callback mechanism. The unique identifiers must be discarded once the results of the RQL requests associated with them have been received.

In some rare cases, the programming scenario at hand may require that the results of an RQL command be received prior to executing any subsequent program statements. In other words, the RQL command needs to be executed in a blocking fashion. To enable this blocking behavior, albeit ill-advised for performance and fault-tolerance reasons, the programmers can simply add the adverb `-blocking` to any RQL command. The runtime processes this directive by finishing the specified command first and returning the results back to the caller.

One peculiarity of BTLE communication is that each device can serve either a peripheral or central role, roughly corresponding to the traditional server/client functionalities, respectively. In other words, the peripheral role entails advertising services, each having potentially multiple characteristics, while the central role entails locating or accessing the services of the devices playing the peripheral role. This clear role separation is currently only supported by iOS devices and Android devices with the latest OS distribution.

To accommodate these distinct roles, the RQL runtime enables mobile devices communicating via BTLE to seamlessly process and advertise resources. The runtime keeps track of the available services by means of unique ids (UUIDs). Since the UUIDs are universal and guaranteed to be globally unique by the BTLE standard, the RQL runtime can easily keep track of the available services and their associated characteristics. These characteristics can have read, write, and notify properties. Reading allows a central device to read a value, writing allows it to write a new value into the characteristic, and notify allows the central device to subscribe to the characteristic's value, so any changes to it will cause a notification update on the central device.

The RQL runtime currently provides five peripheral device's characteristics: GPS, accelerometer, file writing, HTTP request, and facial recognition. As a means of getting updated values, it provides a special-purpose, subscribable result characteristic. As shown in Fig 11. (a), the RQL runtime of an Android device first scans for nearby devices, and then scans for the services they are advertising. If the RQL verb is "pull", the Android device directly reads the GPS readings from the corresponding BLE characteristic on the iOS side. If the RQL verb is "push", the runtime sends a file in chunks of 20 bytes to the file writing characteristic. Otherwise, it writes the RQL command to the BLE service with the unique taskid, and then reads the results back from the BLE result characteristic.

In the runtime of the central device, as shown in Fig 11. (b), the runtime keeps track of the status of surrounding devices and undergoing tasks. Once the runtime receives a RQL request, it enqueues that request into a task queue and returns a task-id immediately. Meanwhile, the device manager periodically scans the Bluetooth advertisements of surrounding devices to discover the provided services. When a task is popped from the task queue, the runtime parses the RQL command to decide on which peripheral device's characteristic it should query or write.

In the runtime of the peripheral device, all received requests are queued up. Each item contains the id of that request and the RQL command associated with it. When a RQL request is received with an adverb defining latency constraints, the request is added to the head of the queue, so as to prioritize its processing. Otherwise, if no adverb is specified, it is added to the end of the queue. When the runtime wants to process a request, it removes a task from the head of the queue, ensures that the task is unexpired, and executes it using the designated service.

## 6 EVALUATION

In this section, we describe how we evaluated various aspects of the reference implementation of RQL, detailed in Section 5. Our evaluation comprises a small user study, various performance/energy efficiency micro-benchmarks, and a robustness assessment of our retrofitting approach.

**Table 2: Lines of Code**

| | Runtime Based | Built from scratch |
|---|---|---|
| GPS request | 20 | 370 |
| HTTP request | 20 | 556 |
| Facial Recognition | 32 | 883 |

### 6.1 Programmability

First, we evaluated the software engineering benefits of our programming model. To that end, we compared two different implementations of the same resource-sharing scenario: original with all resource sharing functionality implemented from scratch and RQL-based with the major functionality provided by the RQL runtime. In Table 2, for each implementation, we report the total lines of uncommented code (ULOC).

As one can see, using RQL reduces the amount of code the programmer has to write by a factor ranging between 20 and 28. Considering that the written code involves complex asynchronous, distributed processing, this code size reduction is likely to have a high positive impact on the code quality.

To empirically assess how well RQL can assist the programmer in putting in place the inter-device resource sharing functionality, we conducted a user study. To that end, we recruited 10 Junior to Senior level Computer Science students from an intermediate Android development class at Virginia Tech. We divided the recruited students into 2 groups, the experimental and control groups, for novice and experienced Android developers, respectively. The experimental group comprised 6 students with no prior experience in Android programming, while the control group comprised 4 students with several years of Android development experience.

In the beginning, we briefly introduced the concepts of AIDL services, broadcast receivers, and Bluetooth LE. Then, each group was given 90 minutes to complete the programming task of obtaining the GPS sensor reading from an iOS device to an Android device. The experimental group was asked to use RQL, while the control group was asked to use any existing, mainstream Android API. The control group was also given an Android chat sample application as an example from which to draw device-to-device coding idioms.

Table 3 presents the results of the study. To our surprise, none of the students in the control group were able to complete the task successfully, which demonstrates the non-trivial nature of device-to-device communication. The results of the experimental group, armed with RQL, were mixed, with 3 students successfully completing the task, with the remaining 3 giving up before the experiment concluded. Because the group using RQL comprised non-experienced Android programmers, the results above indicate that our programming abstraction provide value by streamlining the process of implementing device-to-device interactions and can become a pragmatic tool for future applications.

Table 3: Study Results

| | | |
|---|---|---|
| Group | 1 | 2 |
| Familiarity with Android Development | Beginner | Familiar |
| Number of students | 6 | 4 |
| Number of students completed the task | 3 | 0 |

Table 4: Energy Consumption per Second

| Status | Energy (mA) | Status | Energy (mA) |
|---|---|---|---|
| ScreenOn | 100 | | |
| BluetoothOn | 1 | BluetoothActive | 66 |
| CpuIdle | 92 | CpuActive | 242 |
| WiFiOn | 6 | WiFiActive | 102 |
| GpsOn | 60 | GpsActive | 300 |
| 3GOn | 10 | 3GActive | 250 |

## 6.2 Experiment Setup

The hardware setup for the following experiments include 4 Android mobile devices (1.5GHz dual-core CPU, 2GB RAM) used as source devices, and 2 iOS devices (1 iPhone 6 and an iPad mini) used as target devices.

To evaluate the energy consumption of these devices, we recorded the execution time between “Start" and “Stop" tags, adding tags for actions, such as “Screen On", “Bluetooth On", “Bluetooth Active", “3G Active", “GPS Active", “CPU Idle/Active” etc. Table 4 shows the manufacturer provided values for energy consumption of these operations. For all graphs, we refer to 'local' and 'remote' meaning requests processed on the user's local device and some external nearby device, respectively.

## 6.3 Local and Remote Energy Evaluation

First, we examine the motivating examples' performance in terms of the energy usage in both the local API calls and the corresponding remote RQL calls. Figure 12 shows the energy used by 100 identical RQL requests on the same and across different devices, respectively. Because of the vastly different energy consumption levels between sensor data and heavy HTTP requests, we use both linear and logarithmic vertical scales to present the results.

The graphs show that, excluding some outliers, both local and remote RQL calls consume energy consistently throughout the experiments. The baseline of both figures is identical and essentially shows how an idle application would be consuming energy. In both local and remote calls, the GPS sensor retrieval consumes far less energy than either of HTTP requests or Facial recognition. To compare various protocols, we also benchmark a “Heavy” HTTP request, representative of work-intensive web-based processing. Given the extensible nature of the RQL runtime, one can easily add emerging communication mechanisms, which can outperform BTLE when executing heavy HTTP requests or other high-throughput processes.

Because communicating with nearby devices consumes additional energy, local RQL calls increase their energy efficiency when processing small loads of requests. However, for requests that can be distributed across several available devices, both energy costs and processing latencies decrease precipitously. Figure 12 also reveals cache correspondences between the same device, primarily for sensor data (GPS). Thus reading the GPS data incurs a single large, upfront cost of connecting to the device, but internally optimizes the subsequent request via the assumption that the GPS readings have not changed. This internal optimization explains the plummet in energy costs of accessing remote sensor data, such as GPS.

## 6.4 Local and Remote RQL Latency Experiments

Consider Figure 13 that shows local and remote latency, respectively. These two graphs demonstrate an important practical advantage of accessing resource of nearby devices. When examining GPS, latency drops steeply similar to energy in the previous section, after incurring the upfront cost of connection. This amortization of initial connect requests ensures far better median latency for these remote calls. In fact, we see that for a computationally intensive operation, such as Facial Recognition, the latency is smaller in remote RQL calls by a factor of nearly 1.4 for only a small request size. If we consider sending large requests for Facial Recognition across even a small subset of nearby devices (say only 3 external devices), the resulting latency reduction far outweighs the additional energy use incurred across all devices in use.

Figure 14 presents a full comparison of median energy and latency measurements. This graph supports our initial assumption about the trade off in energy for decreased latency when processing various request types. It is clear that the only outlier is processing HTTP requests remotely. Given the nature of BTLE small packet transmission size restriction, we observe a larger latency since each piece of the HTTP request is broken up and sent individually. Referring back to one of our motivating examples, consider the traveler to a foreign country who is unable to access local mobile data towers. Providing this functionality to the end user is important irrespective of the resulting performance, as long as it is not prohibitively poor. In other words, not outperforming local requests is a minor hindrance in comparison with not being able to process any requests at all. Nevertheless, this limitation of BTLE motivated our efforts to optimize the RQL runtime.

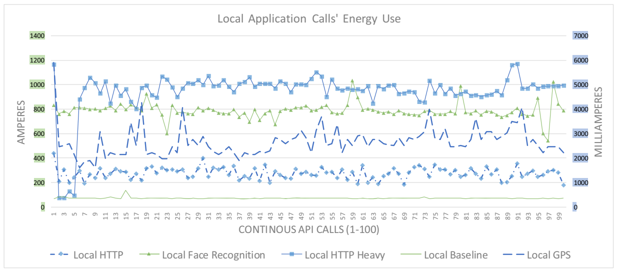
## 7 RELATED WORK

Using nearby mobile devices to cooperatively implement new functionality was originally proposed as a means of exchanging private information over devices for data sharing and data mining [23]. Subsequent research took user mobility into account [19, 24, 35].

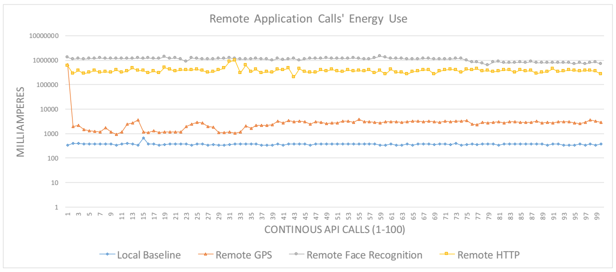
Besides data sharing, another avenue for device cooperation is running map reduce [47] on mobile devices to execute computational-intensive tasks [9, 29, 42]. These approaches, however, are oblivious to device mobility and the preference of users to participate.

In addition to traditional mobile devices, the IoT setups can provide resources for device-to-device resource sharing. Computational tasks have been offloaded to such setups (e.g., Road Side Unit) [16], while mobile messages have been stored and forwarded by a wall-mounted Estimote device [3]. The proposed project will focus on the software engineering aspects of mobile device cooperation, thus benefiting the implementation practices of many of the prior state-of-the-art approaches.

Traditional middleware has been adapted for peer-to-peer resource sharing, including Open CORBA[27], Globe[45] and JXTA[15],

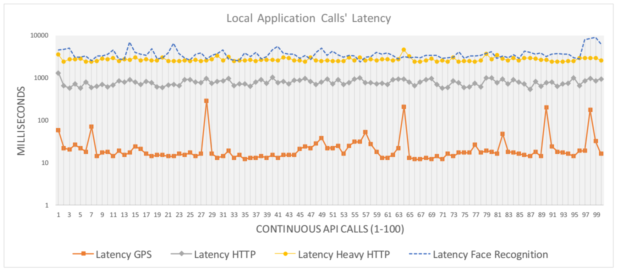


(a) Local Energy Tests

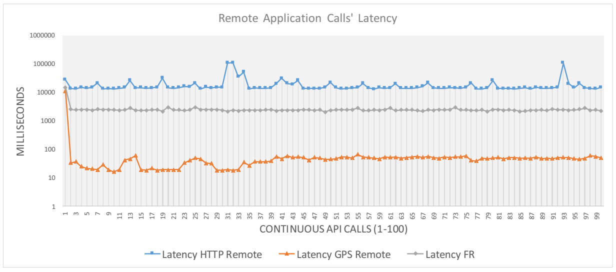


(b) Remote Energy Tests

**Figure 12: Various local and remote RQL command energy usage**



(a) Local Latency Tests



(b) Remote Latency Tests

**Figure 13: Various local and remote request latency use**

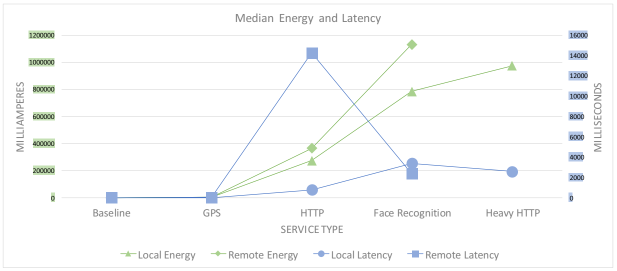


**Figure 14: Median Energy and Latency Across Various Requests**

although without taking device mobility into account. The UPnP protocol[43] enables network devices to provide service to other devices in the network.

Device mobility-aware peer-to-peer resource sharing has started from content sharing [36], with numerous subsequent approaches [2, 6, 10, 20, 21, 32, 33, 40]. Special purpose middleware support face-to-face interactions [41] and cooperative display[4]. These middleware approaches are platform-specific and require modifications at the system level. By contrast, the proposed project aims at heterogeneous device-to-device applications running on top of unmodified system stacks.

The MANET project leverages assistance from devices through multi-hop wireless communication [7]. Various middleware approaches have focused on various aspects of inter-device cooperation, including LIME[31], TOTA[28], Limone[13], CAST[39], MESHmdl[17], Preom [22], MobiPeer [5], Peer2Me [46], Steam [30], Transhumance [37], QAM [14], and MobiCross [8]. These middleware approaches provide programming to control network topologies, network traffic, peer management, etc. By contrast, the proposed approach focuses on supporting mobile application programmers, who are primarily concerned with obtaining the hardware resources they need for their applications.

To support platform independence, [38] proposed using an HTTP server. By contrast, this project focuses on P2P communication, thus reducing communication latencies and processing overhead.

## 8 CONCLUSION

This research focuses on the problem of engineering seamless resource sharing among nearby mobile devices to improve the performance, energy consumption and latency of mobile applications. Although there have been many research publications that have focused on using cooperative device resource sharing to enable new functionalities or to optimize energy and performance, there has not been a push towards software engineering support for application developers to leverage shared resources between heterogeneous mobile devices. To address this problem, we first studied the requirements for leveraging nearby resources in terms of API calls, and then proposed a domain-specific declarative language and a runtime support on two major mobile platforms to enable resource sharing. The results of our case study, user study and efficiency evaluation indicate that our programming model and runtime support can work as a bridge among nearby heterogeneous mobile devices to both improve the programmers' productivity, and optimize the energy consumption and latency of mobile applications. By facilitating the process of implementing cooperative resource sharing among devices, we hope to be able to add this support in the standard toolset for mobile application developers.

## ACKNOWLEDGMENT

This research is supported by the National Science Foundation through the Grants CCF-1717065 and CCF-1649583.

## REFERENCES

[1] Saeid Abolfazli, Zohreh Sanaei, Ejaz Ahmed, Abdullah Gani, and Rajkumar Buyya. 2014. Cloud-based augmentation for mobile devices: motivation, taxonomies, and open challenges. *Communications Surveys & Tutorials, IEEE* 16, 1 (2014), 337–368.

[2] Ardalan Amiri Sani, Kevin Boos, Min Hong Yun, and Lin Zhong. 2014. Rio: a system solution for sharing i/o between mobile systems. In *Proceedings of the 12th annual international conference on Mobile systems, applications, and services (MobiSys'14)*. ACM, 259–272.

[3] Fehmi Ben Abdesslem and Anders Lindgren. 2014. Demo: mobile opportunistic system for experience sharing (MOSES) in indoor exhibitions. In *Proceedings of the 20th annual international conference on Mobile computing and networking (MobiCom'14)*. ACM, 267–270.

[4] Christian Berkhoff, Sergio F Ochoa, José A Pino, Jesus Favela, Jonice Oliveira, and Luis A Guerrero. 2014. Clairvoyance: A framework to integrate shared displays and mobile computing devices. *Future Generation Computer Systems* 34 (2014), 190–200.

[5] Mario Bisignano, Giuseppe Di Modica, and Orazio Tomarchio. 2005. JMobiPeer: a middleware for mobile peer-to-peer computing in MANETs. In *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems Workshops (ICDCS'05 Workshop)*. IEEE, 785–791.

[6] Mauro Caporuscio, P-G Raverdy, and Valerie Issarny. 2012. ubiSOAP: A service-oriented middleware for ubiquitous networking. *Services Computing, IEEE Transactions on* 5, 1 (2012), 86–98.

[7] Eduardo da Silva and Luiz Carlos P Albini. 2014. Middleware proposals for mobile ad hoc networks. *Journal of Network and Computer Applications* 43 (2014), 103–120.

[8] Mieso K Denko, Elhadi Shakshuki, and Haroon Malik. 2007. A mobility-aware and cross-layer based middleware for mobile ad hoc networks. In *Proceedings of the 21st International Conference on Advanced Information Networking and Applications (AINA'07)*. IEEE, 474–481.

[9] Adam Dou, Vana Kalogeraki, Dimitrios Gunopulos, Taneli Mielikainen, and Ville H Tuulos. 2010. Misco: a MapReduce framework for mobile systems. In *Proceedings of the 3rd International Conference on Pervasive Technologies related to Assistive Environments*. ACM, 32.

[10] Daniel J Dubois, Yosuke Bando, Konosuke Watanabe, and Henry Holtzman. 2013. ShAir: Extensible middleware for mobile peer-to-peer resource sharing. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering (FSE'13)*. ACM, 687–690.

[11] John Edstrom and Eli Tilevich. 2014. Improving the survivability of RESTful Web applications via declarative fault tolerance. *Concurrency and Computation: Practice and Experience* (2014), n/a–n/a. DOI: http://dx.doi.org/10.1002/cpe.3197

[12] Roy Thomas Fielding. 2000. *Architectural styles and the design of network-based software architectures*. Ph.D. Dissertation. University of California, Irvine.

[13] Chien-Liang Fok, Gruia-Catalin Roman, and Gregory Hackmann. 2004. A lightweight coordination middleware for mobile computing. In *Coordination Models and Languages*. Springer, 135–151.

[14] Abhrajit Ghosh, Shih-wei Li, C Jason Chiang, Ritu Chadha, Kimberly Moeltner, Syeed Ali, Yogeeta Kumar, and Rocio Bauer. 2010. QoS-aware Adaptive Middleware (QAM) for tactical MANET applications. In *MILCOM'10*. IEEE, 178–183.

[15] Li Gong. 2001. JXTA: A network programming environment. *Internet Computing, IEEE* 5, 3 (2001), 88–95.

[16] Kiryong Ha, Zhuo Chen, Wenlu Hu, Wolfgang Richter, Padmanabhan Pillai, and Mahadev Satyanarayanan. 2013. *Towards wearable cognitive assistance*. Technical Report. DTIC Document.

[17] Klaus Herrmann. 2003. Meshmd1-a middleware for self-organization in ad hoc networks. In *ICDCS'03 Workshop*. IEEE, 446–451.

[18] Mohammad Ashraful Hoque, Matti Siekkinen, and Jukka K Nurminen. 2014. Energy efficient multimedia streaming to mobile devicesâĂŤa survey. *Communications Surveys & Tutorials, IEEE* 16, 1 (2014), 579–597.

[19] Pan Hui, Augustin Chaintreau, James Scott, Richard Gass, Jon Crowcroft, and Christophe Diot. 2005. Pocket switched networks and human mobility in conference environments. In *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking (WDTN'05)*. ACM, 244–251.

[20] Peng Jiang, John Bigham, Eliane Bodanese, and Emmanuel Claudel. 2011. Publish/subscribe delay-tolerant message-oriented middleware for resilient communication. *Communications Magazine, IEEE* 49, 9 (2011), 124–130.

[21] David Koll, Jun Li, and Xiaoming Fu. 2014. SOUP: an online social network by the people, for the people. In *Proceedings of the 2014 ACM conference on SIGCOMM*. ACM, 143–144.

[22] Gerd Kortuem. 2002. Proem: a middleware platform for mobile peer-to-peer computing. *ACM SIGMOBILE Mobile Computing and Communications Review* 6, 4 (2002), 62–64.

[23] Gerd Kortuem, Jay Schneider, Dustin Preuitt, Thaddeus G Cowan Thompson, Stephen Fickas, and Zary Segall. 2001. When peer-to-peer comes face-to-face: Collaborative peer-to-peer computing in mobile ad-hoc networks. In *Proceedings of 1st International Conference on Peer-to-Peer Computing (P2P'01)*. IEEE, 75–91.

[24] Niko Kotilainen, Matthieu Weber, Mikko Vapa, and Juori Vuori. 2005. Mobile Chedar-a peer-to-peer middleware for mobile devices. In *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom'05 WorkShop)*. IEEE, 86–90.

[25] Young-Woo Kwon and Eli Tilevich. 2012. Energy-efficient and fault-tolerant distributed mobile execution. In *Proceedings of the* $32^{th}$ *IEEE International Conference on Distributed Computing Systems (ICDCS'12)*. IEEE, 586–595.

[26] Young-Woo Kwon and Eli Tilevich. 2014. Cloud refactoring: automated transitioning to cloud-based services. *Automated Software Engineering* 21, 3 (2014), 345–372.

[27] Chaoying Ma and Jean Bacon. 1998. COBEA: A CORBA-based event architecture. In *Proceedings of the 4th conference on USENIX Conference on Object-Oriented Technologies and Systems-Volume 4*. USENIX Association, 9–9.

[28] Marco Mamei, Franco Zambonelli, and Letizia Leonardi. 2003. Tuples on the air: A middleware for context-aware computing in dynamic networks. In *Proceedings of 23rd International Conference on Distributed Computing Systems Workshops (ICDCS'03 Workshop)*. IEEE, 342–347.

[29] Eugene E Marinelli. 2009. *Hyrax: cloud computing on mobile devices using MapReduce*. Technical Report. DTIC Document.

[30] René Meier and Vinny Cahill. 2002. Steam: Event-based middleware for wireless ad hoc networks. In *ICDCS'02 Workshop*. IEEE, 639–644.

[31] Amy L Murphy, Gian Pietro Picco, and Gruia-Catalin Roman. 2006. LIME: A coordination model and middleware supporting mobility of hosts and agents. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 15, 3 (2006), 279–328.

[32] Kazuhiro Nakao and Yukikazu Nakamoto. 2012. Toward remote service invocation in android. In *Proceedings of the 9th International Conference on Ubiquitous Intelligence & Computing and 9th International Conference on Autonomic & Trusted Computing (UIC'12)*. IEEE, 612–617.

[33] Andrés Neyem, Sergio F Ochoa, José A Pino, and Rubén Darío Franco. 2012. A reusable structural design for mobile collaborative applications. *Journal of Systems and Software* 85, 3 (2012), 511–524.

[34] Cătălin Nicutar, Dragoş Niculescu, and Costin Raiciu. 2014. Using Cooperation for Low Power Low Latency Cellular Connectivity. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*. ACM, 337–348.

[35] Jörg Ott, Esa Hyytia, Pasi Lassila, Tobias Vaegs, and Jussi Kangasharju. 2011. Floating content: Information sharing in urban areas. In *Proceedings of the 2011 IEEE International Conference on Pervasive Computing and Communications (PerCom'11)*. IEEE, 136–146.

[36] Maria Papadopouli and Henning Schulzrinne. 2001. Design and implementation of a peer-to-peer data dissemination and prefetching tool for mobile users. In *Proceedings of the first NY Metro Area Networking workshop (NYMAN'01)*.

[37] Guilhem Paroux, Ludovic Martin, Julien Nowalczyk, and Isabelle Demeure. 2007. Transhumance: A power sensitive middleware for data sharing on mobile ad hoc networks. In *Proceedings of the 7th international Workshop on Applications and Services in Wireless Networks (ASWN'07)*.

[38] Pierluigi Plebani, Cinzia Cappiello, Marco Comuzzi, Barbara Pernici, and Sandeep Yadav. 2012. MicroMAIS: executing and orchestrating Web services on constrained mobile devices. *Software: Practice and Experience* 42, 9 (2012), 1075–1094.

[39] Gruia-Catalin Roman, Radu Handorean, and Rohan Sen. 2006. Tuple space coordination across space and time. In *Coordination Models and Languages*. Springer, 266–280.

[40] Ahmed Salem and Tamer Nadeem. 2014. Colphone: A smartphone is just a piece of the puzzle. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*. ACM, 263–266.

[41] Genaro Saucedo-Tejada, Sonia Mendoza, and Dominique Decouchant. 2013. F2FMI: A toolkit for facilitating face-to-face mobile interaction. *Expert Systems with Applications* 40, 15 (2013), 6173–6184.

[42] Cong Shi, Vasileios Lakafosis, Mostafa H Ammar, and Ellen W Zegura. 2012. Serendipity: enabling remote computing among intermittently connected mobile devices. In *Proceedings of the thirteenth ACM international symposium on Mobile Ad Hoc Networking and Computing (MobiHoc'12)*. ACM, 145–154.

[43] Hyungjoo Song, Daeyoung Kim, Kangwoo Lee, and Jongwoo Sung. 2005. UPnP-based sensor network management architecture. In *Proc. International Conference on Mobile Computing and Ubiquitous Networking*.

[44] Eli Tilevich and Young-Woo Kwon. 2014. Cloud-based execution to improve mobile application energy efficiency. *Computer* 47, 1 (2014), 75–77.

[45] Maarten Van Steen, Philip Homburg, and Andrew S Tanenbaum. 1999. Globe: A wide-area distributed system. *IEEE concurrency* 7, 1 (1999), 70–78.

[46] Alf Inge Wang, Tommy Bjornsgard, and Kim Saxlund. 2007. Peer2me-rapid application framework for mobile peer-to-peer applications. In *Proceedings of the*

*2007 International Symposium on Collaborative Technologies and Systems (CTS'07).* IEEE, 379–388.

[47] Hung-chih Yang, Ali Dasdan, Ruey-Lung Hsiao, and D Stott Parker. 2007. Map-reduce-merge: simplified relational data processing on large clusters. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data.* ACM, 1029–1040.

[48] Pengfei Zhou, Mo Li, and Guobin Shen. 2014. Use it free: Instantly knowing your phone attitude. In *Proceedings of the 20th annual international conference on Mobile computing and networking.* ACM, 605–616.