Wireless Netw (2014) 20:361–378
DOI 10.1007/s11276-013-0608-7

Energy-efficient and cost-effective web API invocations with transfer size reduction for mobile mashup applications

Chen-Che Huang · Jiun-Long Huang · Chin-Liang Tsai · Guan-Zhong Wu · Chia-Min Chen · Wang-Chien Lee

Published online: 20 June 2013

Abstract Recently, the proliferation of smartphones and the extensive coverage of wireless networks have enabled numerous mobile users to access Web resources with smartphones. Mobile mashup applications are very attractive to smartphone users due to specialized services and user-friendly GUIs. However, to offer new services through the integration of Web resources via Web API invocations, mobile mashup applications suffer from high energy consumption and long response time. In this paper, we propose a proxy system and two techniques to reduce the size of data transfer, thereby enabling mobile mashup applications to achieve energy-efficient and cost-effective Web API invocations. Specifically, we design an API query language that allows mobile mashup applications to readily specify and obtain desired information by instructing a proxy to filter unnecessary information returned from Web API servers. We also devise an image multi-get module, which results in mobile mashup applications with smaller transfer sizes by combining multiple images and adjusting the quality, scale, or resolution of the images. With the proposed proxy and techniques, a mobile mashup application can rapidly retrieve Web resources via Web API invocations with lower energy consumption due to a smaller number of HTTP requests and responses as well as smaller response bodies. Experimental results show that the proposed proxy system and techniques significantly reduce transfer size, response time, and energy consumption of mobile mashup applications.

Keywords Mobile mashup applications · Web resource · Web API · Energy efficiency · Transfer size reduction · Smartphones

C.-C. Huang · J.-L. Huang (✉) · C.-L. Tsai · G.-Z. Wu · C.-M. Chen
Department of Computer Science, National Chiao Tung University, Hsinchu, Taiwan, ROC
e-mail: jlhuang@cs.nctu.edu.tw

W.-C. Lee
Department of Computer Science and Engineering, The Pennsylvania State University, University Park, PA 16802, USA
e-mail: wlee@cse.psu.edu

1 Introduction

1.1 Background

In recent years, the ubiquity of smartphones and the widespread deployment of wireless networks have resulted in more people using smartphones to access Web resources [14]. Contrary to Web browsing by computer users, smartphone users prefer utilizing mobile applications to access Web services. By exploiting the advanced capabilities of smartphones, mobile applications are able to provide intricate GUIs and can be tailored to meet the specific needs of the users. Instead of visiting one particular Web site, mobile mashup applications create new and specialized applications by integrating Web resources of various Web sites [20]. Specifically, mobile mashup applications are able to aggregate a variety of Web resources by invoking the Web APIs released by well-known Web sites, such as Google, Yahoo!, Facebook, and YouTube. A good example of a mobile mashup application is HTC Friend Stream, which aggregates up-to-date information about the friends of a smartphone user from various social networking sites (i.e., Facebook, Twitter, and Plurk), as shown in Fig. 1. With Friend Stream, a mobile user is able to easily receive the latest status of his/her friends without separately browsing these social networking sites.

Fig. 1 A screenshot of HTC Friend Stream

Despite the benefits of mobile mashup applications, Web API invocations are usually energy- and time-consuming for smartphones. Because the battery life of smartphones remains constrained and the long response times easily discourage mobile users [13], achieving fast and energy-efficient Web API invocations is crucial to mobile mashup applications. Shye et al. [19] reported in that the wireless interface is one of the most energy-consuming components of a smartphone; this realization motivates us to attempt to reduce energy consumption and response time by reducing transfer sizes between mobile applications and Web API servers. In addition, many users continue to subscribe to usage-based pricing plans especially in developing regions [3]. Therefore, reducing transfer size for Web resource retrieval will also reduce the cost for these users. To the best of our knowledge, no prior work has addressed the issue of transfer size reduction for mobile mashup applications.

1.2 Observations

To better understand data transfer between smartphones and Web API servers, we first analyze the data transfer incurred by Web API invocations because invoking Web APIs is the essence of mobile mashup applications. To facilitate our analysis, we implement several mobile mashup applications on top of the Google Android emulator and use Wireshark[1] to capture and study the packets among the implemented mashup mobile applications and the Web API servers. Considering the trend of and interests in mobile mashup applications, we select the Web APIs provided by social networking sites (Facebook and Twitter), Web album sites (Picasa and Flickr), video sharing sites (YouTube) and information provision sites (Yahoo! Lifestyle). The Web APIs that are used are summarized in Table 1. In the following, we elaborate our four main observations.

- **Observation 1: Excessive HTTP requests and responses for API resource retrieval**.
  When a mobile mashup application extracts Web resources from one Web API server, it has to invoke the corresponding Web API method by sending an HTTP request to the Web API server; the Web API server will return at least one HTTP response. Multiple HTTP requests and responses are required for retrieving resources from multiple Web API servers. Consider HTC Friend Stream mentioned previously as an example. To integrate the latest published messages of friends from several social networking sites, the application invokes multiple Web APIs by issuing multiple HTTP requests to these Web API servers and then obtains the results via multiple HTTP responses returned from these Web API servers, as depicted in Fig. 2. Such excessive HTTP requests and responses create substantial overhead for a mobile mashup application, especially considering that each HTTP request contains only one Web API invocation but incurs the HTTP header overhead. This observation motivates us to design mechanisms that allow a mobile mashup application to invoke multiple Web APIs with only one HTTP request.
- **Observation 2: Parameter dependency of Web APIs**.
  To facilitate resource management, Web resources are often organized into hierarchical structures. However, such structures may cause dependency of parameters of Web APIs. For instance, prior to listing the districts of Taipei city by using Yahoo! Lifestyle APIs, the city *ID* of Taipei is required because the resulting list of districts depends on the city *ID*. Specifically, we have to first call the Web API "*Addr.listCity*" to obtain the city *ID* of Taipei, which is designated "A" here. Subsequently, the replied *ID* allows us to obtain district information by invoking the Web API "*Addr.listDistrict?city=A*" (see Fig. 3). Similarly, for YouTube APIs, we have to retrieve the video ID prior to accessing the desired videos. In these cases, the dependency of parameters of Web APIs leads to an additional pair of HTTP requests and responses. In our experiment in Sect. 4.2, the overhead in transfer size is about 43.2–88.4 %.

[1] http://www.wireshark.com

Table 1 Observed web APIs

Web site	Web API	Associated observations
Yahoo!	http://tw.lifestyle.yahooapis.com/v0.3/Addr.listDistrict	1, 2
YouTube	http://gdata.youtube.com/feeds/projection/videos	1, 2
Facebook	https://graph.facebook.com/me/home	1, 3
Twitter	https://api.twitter.com/1/statuses/update.xml	1, 3
Picasa	https://picasaweb.google.com/data/feed/api/user/userID/albumid/albumID	1, 4
Flickr	http://www.flickr.com/services/api/explore/flickr.galleries.getPhotos	1, 4

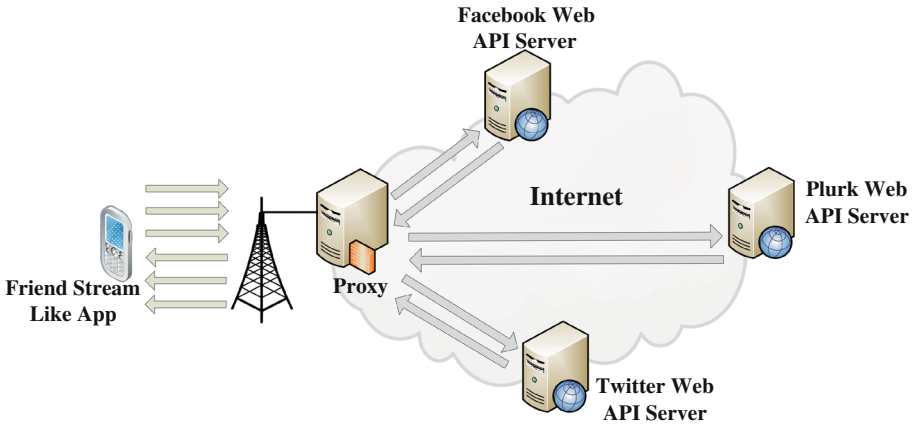


Fig. 2 Multiple HTTP requests and responses for HTC Friend Stream-like application

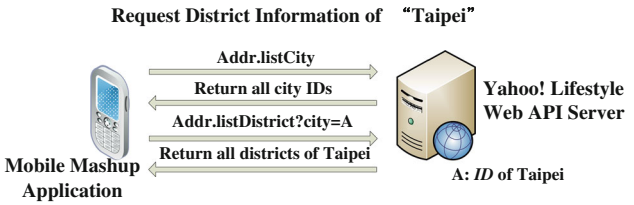


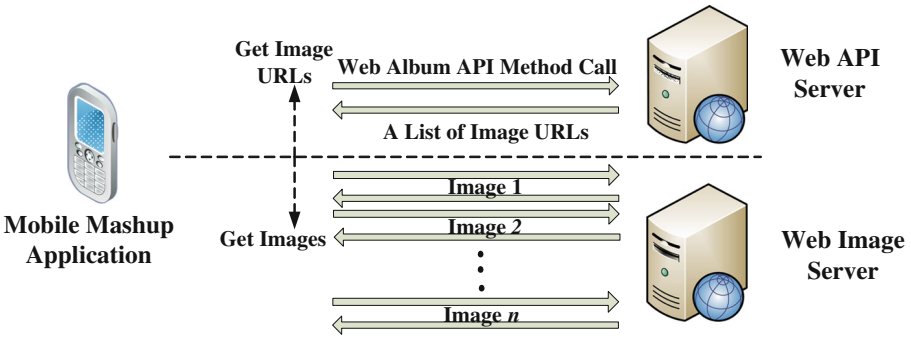
Fig. 3 An example of parameter dependency of Web APIs

- **Observation 3: Unnecessary information in an HTTP response**.
  Because Web API servers are unable to anticipate the exact needs of mobile mashup applications, their Web APIs usually return unnecessary/overly-detailed information to mashup applications. Consider Twitter API as an example. When a mobile mashup application wants to extract the names and nicknames of a user's friends on Twitter, it calls the Web API "https://api.twitter.com/1/statuses/home_timeline.xml?include_entities=true".
  In addition to the desired names and nicknames, the response also contains unnecessary information such as background and text colors of the user's Twitter page (refer to Fig. 4). In our experiment in Sect. 4.1, about 88.4 % of information is unnecessary for our mobile mashup applications. Therefore, we propose enabling mobile mashup applications to specify the information of interest to filter unnecessary information.
- **Observation 4: Inflexible Web APIs for images and multiple HTTP requests for multi-image retrieval**.
  Figure 5 illustrates the HTTP requests and responses when downloading album images by calling a typical Web album API. Rather than the images of interest, the Web API server returns the corresponding URLs of the images in the album. Thus, for a Web album that consists of n images, $n + 1$ HTTP requests and $n + 1$ HTTP responses have to be issued to acquire those images, incurring a significant transfer cost. We also make two more observations. First, we find that the returned images are usually identical to the original user-uploaded images, which are typically much larger than the display areas assigned by mobile mashup applications. This causes a mobile mashup application to suffer from large transfer sizes and long response times. Unfortunately, most Web album APIs do not allow mobile mashup applications to specify the desired quality, scale or resolution. Second, the images are transmitted separately and the album APIs do not support batch image retrieval. Retrieving multiple images in a batch manner is beneficial due to the reduction in HTTP header overhead (23–34 % overhead in our experiment in Sect. 4.3). In addition to Web album APIs, image retrieval APIs for other types of

Fig. 4 The result returned by the Twitter API "*home_timeline*"

```
<?xml version="1.0" encoding="UTF-8"?>
<statuses type="array">
<status>
  <created_at>Thu Aug 18 14:47:28 +0000 2011</created_at>
  <id>104202760567791618</id>
  <text>An apple a day keeps the doctor away</text>
  <source>web</source>
  <truncated>false</truncated>
  <favorited>false</favorited>
  <in_reply_to_status_id></in_reply_to_status_id>
  <in_reply_to_user_id></in_reply_to_user_id>
  <in_reply_to_screen_name></in_reply_to_screen_name>
  <retweet_count>0</retweet_count>
  <retweeted>false</retweeted>
  <user>
    <id>262165567</id>
    <name>Chenjm</name>
    <screen_name>Chenjm</screen_name>
    <location></location>
    <description></description>
    <profile_image_url>http://a1.twimg.com/sticky/default_profile_images/default_profi
le_6_normal.png</profile_image_url>
    <profile_image_url_https>https://si0.twimg.com/sticky/default_profile_images/defau
lt_profile_6_normal.png</profile_image_url_https>
    <url></url>
    <protected>false</protected>
    <followers_count>1</followers_count>
    <profile_background_color>C0DEED</profile_background_color>
    <profile_text_color>333333</profile_text_color>
    <profile_link_color>0084B4</profile_link_color>
              .
              .
              .
  </user>
  <geo/>
  <coordinates/>
  <place/>
  <contributors/>
</status>
```

Fig. 5 Retrieve album photos by calling Web album APIs



Web sites (e.g., Facebook and Twitter) have similar issues. We develop a flexible image retrieval mechanism for mobile mashup applications so that these applications can receive images satisfying their criteria at a lower transmission cost by performing image adjustment and combination in the proxy.

To summarize, mobile mashup applications are impaired by large transfer sizes when aggregating Web resources of interest via Web API invocations. Large transfer sizes cause significant energy consumption and response time for mobile mashup applications, and high costs for subscribers of usage-based pricing plans. To enable fast, energy-efficient, and cost-effective Web API invocations, we aim to reduce the transfer size in four facets as follows: (1) reduce the number of HTTP requests and responses; (2) eliminate the extra Web API invocations caused by parameter dependency; (3) filter unnecessary information of HTTP responses; (4) enable the proxy-side adjustment of quality, scale and resolution of images, as well as batch image retrieval.

1.3 Our contributions

In this paper, we propose a proxy system and several techniques to reduce the transfer size of mobile mashup applications by addressing the observed problems. In the proposed system, a proxy is deployed on the Internet and serves as an intermediate node between mobile mashup applications and Web API servers. The proxy and mobile mashup applications are implemented with the following techniques.

- API Query Language (AQL). To eliminate the problem of unnecessary information (Observation 3) of Web API invocations, we propose the use of AQL to enable mobile mashup applications to specify the information of interest with SQL-like instructions. With AQL, the proxy is able to filter unnecessary information and return only the desired information to mobile mashup applications, thereby reducing HTTP response sizes. Mobile mashup applications can also invoke multiple Web APIs by sending one AQL message that contains multiple AQL instructions to the proxy. Thus, the number of HTTP requests and responses can be greatly reduced (Observation 1). In addition, AQL also enables mobile mashup applications to describe the dependency of successive Web API invocations. By doing so, mobile mashup applications are able to ask the proposed proxy system to invoke multiple Web APIs with dependency of behalf of them, thereby overcoming the problem resulting from parameter dependency of Web APIs (Observation 2).
- Image Multi-Get (IMG) Module. To make retrieved images compatible with mobile mashup applications (Observation 4), we present the IMG module for proxy-side adjustment in quality, scale and resolution of images. Also, the IMG module provides image combination, allowing the proxy to transmit images in a batch manner to alleviate the overhead of multiple HTTP requests and responses (Observation 1).

To take advantage of AQL and the IMG module, developers of mobile mashup application have to implement mobile mashup applications to retrieve Web resources via AQL and IMG instructions rather than direct Web API invocations. With the proposed proxy and techniques, when intending to retrieve some Web API resources, a mobile mashup application puts corresponding AQL or IMG instructions in a single HTTP request to the proxy. According to the received AQL and/or IMG instructions, the proxy then extracts the desired resources and filters and/or adjusts the resources, so that the application will receive only the exactly interesting parts of the resources. With the proxy together with the proposed techniques, a mobile mashup application benefits from substantial transfer size reduction since fewer HTTP requests and responses as well as smaller response sizes are incurred. As a result, the application can obtain the desired information rapidly with less energy consumption. For performance evaluation, we have implemented a prototype of the proposed proxy system and conducted experiments using a smartphone over a Wi-Fi network. A wide range of Web APIs are employed and several mashup applications are created to measure the performance. The extensive experimental results show that the proposed system and techniques are able to significantly reduce the transfer size, thereby increasing battery life and shortening the response time for mobile mashup applications.

The rest of this paper is organized as follows. Section 2 reviews the prior work related to transfer size reduction. In Sect. 3, we describe the proposed proxy system and elaborate the techniques for transfer size reduction. The experimental results are presented in Sect. 4. Finally, Sect. 5 concludes this paper and presents our future work.

2 Related work

Since the first generation of smartphones was released, smartphones have been accepted at a blistering pace. Because of their immense popularity, smartphones have received considerable attention from the research community [2, 5, 6, 12, 18]. Since battery life is crucial for their operation and remains limited, a significant number of studies have focused on energy savings for smartphones [16, 17]. Some previous studies have concentrated on reduced energy consumption for different smartphone components such as user interface [22], wireless interface [4, 8], and backlight [11].

As reported in [19], the wireless interface is one of the most energy-consuming components of a smartphone. To reduce energy consumption, most smartphones will convert the wireless interface (e.g., Wi-Fi, 3G) into doze mode when there is no data transfer; transfer size reduction is an important method for reducing the energy-consumption of smartphones. Han et al. [9] proposed an image transcoding proxy which is able to control response time to meet user requirements by transcoding images into proper resolutions and quality. The proposed transcoding proxy can use the lossy compression method to adaptively adjust the transfer sizes of images. An analytical framework is also proposed to determine whether to transcode an image and how to transcode the image to meet reponse time that is acceptable to the user.

In Housel et al. [10] proposed a client/intercept based approach to reduce the transfer sizes of HTTP requests and responses. With the cooperation of client side intercepts and server side intercepts, the sizes of the headers of HTTP requests and responses can be reduced. Housel et al. also proposed the use of caching and differencing techniques to eliminate unnecessary portions in HTTP responses, which greatly reduces the transfer sizes between mobile clients and Web servers.

Zhao et al. [21] proposed in a virtual machine-based proxy (VMP) approach to reduce the computation overhead and transfer size by shifting the computing of complex Web page retrieval and rendering from smartphones to virtual machines in cloud computing environments. Specifically,

when the Web browser in a smartphone sends a Web page request to the corresponding proxy, the proxy will retrieve all components (e.g., images, flash, javascript) of the requested Web page, render the Web page to an image, and send the image to the smartphone. Experimental results show that the retrieval time and transfer size of complex Web pages can be greatly reduced with VMP. Opera has developed two Web browsers for smarphones, Opera Mobile and Opera Mini [15]. Opera Mobile and Opera Mini are capable of utilizing Opera' servers to compress Web content before it reaches smartphones, thereby reducing the transfer sizes. Google has proposed a new protocol, SPDY [7], for faster Web access. With the proposed techniques such as multiplexed streams and HTTP header compression, SPDY is able to achieve faster Web page load time and smaller transfer size. Amazon has developed a new Web browser, Silk [1], to reduce the Web page load time and energy consumption of mobile devices by offloading some Web content processing (such Web object retrieval and rendering) from mobile devices to AWS (Amazon Web Services) cloud platform.

Although transfer size reduction for mobile devices/smartphones has been well studied, to the best of our knowledge, most previous studies focus on transfer size reduction in Web browsing in mobile environments and no prior work has focused on transfer size reduction for Web API invocations. As the popularity of mobile mashup applications increases, we believe that transfer size reduction for Web API invocations will become a research emphasis.

3 Proposed proxy system

In this section, we first introduce the system architecture and describe how mobile mashup applications can benefit from the proposed proxy system and techniques that can reduce the number of HTTP requests and responses as well as HTTP response sizes. Then, we detail the proposed techniques, namely API query language (AQL) and image multi-get (IMG) module, for mobile mashup applications to reduce transfer size in Web API invocations.

3.1 System architecture

Figure 6 depicts the architecture of the proposed proxy system. The proposed system consists of three components: mobile mashup applications, an intermediate proxy, and Web API servers. An intermediate proxy acts as an aggregator server for mobile mashup applications and is deployed on the Internet. To achieve transfer size reduction, mobile mashup applications and the proxy should be implemented with the proposed AQL and IMG module. With the proposed techniques, mobile mashup applications focus only on specifying the resources of interest. The proxy is responsible for requesting resources from Web API servers by invoking the corresponding Web APIs and processing the resources according to the requirements of the applications. Note that no modification to Web API servers is required in the proposed system, indicating easy deployment of the proposed system.

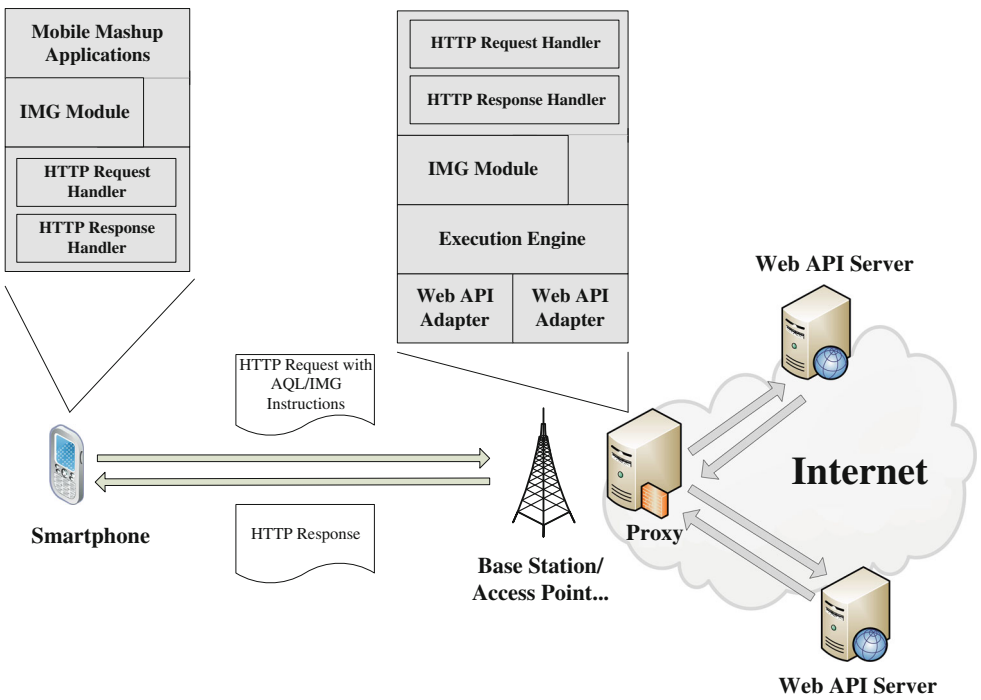


Fig. 6 System architecture

To use the proposed system to reduce transfer size, the mobile mashup applications should utilize AQL and IMG instructions, rather than Web APIs, to retrieve resources of interest. Each AQL or IMG instruction is sent to the proxy as a parameter of an HTTP request. In the proxy, for each Web API server, there is a Web API Adapter to encapsulate all Web APIs provided by the Web API server. The goal of the Web API Adapters is to provide uniform interfaces/data formats for the execution engine. Thus, when receiving an AQL or an IMG instruction, the proxy will use the execution engine to run the instruction with the assistance from the underlying Web API adapters.

The procedure of using AQL and IMG instructions to retrieve Web resources is below.

- *Step 1:* When retrieving Web resources, a mobile mashup application creates an HTTP request and adds the corresponding AQL/IMG instruction to the HTTP request as the body of the HTTP request. To reduce the number of HTTP requests, the application wraps the AQL and/or IMG instructions into a single HTTP request. The mobile mashup application then sends the HTTP request to the proxy-side HTTP request handler via the client-side HTTP request handler.
- *Step 2:* On receiving an HTTP request, the proxy-side HTTP request handler first checks whether the HTTP request is an AQL instruction, an IMG instruction or a normal Web API invocation. When the HTTP request is an AQL instruction or an IMG instruction, the proxy-side HTTP request handler asks the execution engine to handle the HTTP request by Step 2-1 or Step 2-2, respectively. Otherwise, the proxy-side HTTP request handler will execute Step 2-3 to handle the normal Web API invocation.

  – *Step 2-1: (AQL Instruction Handling)*

    * *Step 2-1-1:* When the HTTP request is an AQL instruction, the execution engine extracts the corresponding Web API information from the AQL instruction and invokes these Web API(s) to retrieve resources with the aid of the corresponding Web API Adapter(s). With the uniform data formats provided by these Web API Adapters, these resources are then combined into one message.
    * *Step 2-1-2:* The execution engine then processes the message by removing unnecessary information based on the AQL instruction.
    * *Step 2-1-3:* Finally, the execution engine sends the reduced resources to the proxy-side HTTP response handler as the resource requested by the application.

  – *Step 2-2: (IMG Instruction Handling)*

    * *Step 2-2-1:* When the HTTP request is an IMG instruction, the execution engine extracts the corresponding Web API information from the IMG instruction, and invokes these Web API(s) to obtain the URLs of the desired images with the aid of the corresponding Web API Adapter(s).
    * *Step 2-2-2:* After obtaining these URLs, the execution engine retrieves each image by sending the corresponding HTTP request to the corresponding Web image server.
    * *Step 2-2-3:* When these images are ready, the execution engine asks the proxy-side IMG module to adjust these images based on the IMG instruction and to apply the proposed image combination method (described in Sect. 3.3) to combine these images into one message.
    * *Step 2-2-4:* Finally, the execution engine sends the resultant message to the proxy-side HTTP response handler as the resource requested by the application.

  – *Step 2-3: (Normal Web API Invocation Handling)* When the HTTP request is a normal Web API invocation, the proxy-side HTTP request handler invokes the Web API by issuing a new HTTP request to the corresponding Web API server, and sends the received resource to the proxy-side HTTP response handler as the resource requested by the application.

- *Step 3:* The proxy-side HTTP response handler creates a new HTTP response and adds the resource requested by the application as the body of the HTTP response. The proxy-side HTTP response handler then sends the HTTP response to the corresponding client-side HTTP response handler.
- *Step 4:* When receiving an HTTP response, the client-side HTTP response handler first determines whether the HTTP response is for an IMG instruction. If so, the client-side HTTP response handler asks the client-side IMG module to split the content of the HTTP response into single/multiple image(s), and returns the image(s) to the mobile mashup applications. Otherwise, the client-side HTTP response handler simply returns the content of the HTTP response to the mobile mashup application.

With the assistance of the proxy and the proposed techniques, a mobile mashup application consumes less energy and experiences shorter response time for Web API resource retrieval due to a smaller number of HTTP requests and responses and a compact HTTP response body. For backward compatibility, the proposed system supports existing mobile mashup applications using Web APIs directly by

simply relaying their requests to the corresponding Web API servers, as described in Sep 2-3. Note that these applications can function normally in the proposed system but are unable to benefit from it. In addition, since the proposed proxy is able to forward normal Web API invocations, it is possible for developers to implement mobile mashup applications in a hybrid manner. That is, the developers are able to invoke some heavy Web APIs by AQL/IMG instructions and invoke other light Web APIs by the normal Web API invocations. The above hybrid implementation is able to reduce the cost of re-implementing existing mobile mashup applications to enjoy the advantages of the proposed proxy.

3.2 API query language

To address the problems of unnecessary information and parameter dependency of Web APIs, we design *API Query Language*, abbreviated as AQL, to enable mobile mashup applications to specify the information of interest. Specifically, mobile mashup applications can use AQL instructions to create filters that allow the proxy to remove unnecessary information and to send only information of interest to mashup applications.

The current Web APIs are implemented using four HTTP methods: POST, GET, PUT, and DELETE. In accordance with the implementation, the design of AQL is based on these four HTTP methods. Table 2 displays the format, which represents a Web API invocation into the corresponding AQL instruction. In what follows, we elaborate the usage of the AQL through examples to provide a better understanding of the AQL.

- **Insert:**
  When a mobile mashup application wants to create a new resource on the Web API server, it creates an AQL insert instruction because resource creation is realized using an HTTP POST method. After receiving the AQL insert instruction, the proxy reverts to the original Web API and invokes the Web API for the application. For example, consider that a mobile mashup application wants to create a message "*Tweet*" on the mobile user's Twitter webpage. This application sends the AQL insert instruction
  INSERT INTO https://api.twitter.com/1/statuses/update.xml?status= *VALUES (Tweet)* to the proxy to invoke the Web API
  https://api.twitter.com/1/statuses/update.xml?status=Tweet.
  After receiving this AQL query, the proxy invokes the corresponding Web API to create the new message for the requesting application. Finally, the proxy will reply with the HTTP response received from the API server to the mobile mashup application. To handle large data, a mobile mashup application should use the POST method to send a multipart HTTP request with two parts: the first part is the AQL instruction while the second part is the data.
- **Select**: If a mobile mashup application wants to retrieve the desired information from a Web API server through a Web API invocation, it composes an AQL select instruction because resource retrieval is implemented using an HTTP GET method. As mentioned in Observation 3, a Web API server may return the result with unnecessary information. Thus, the AQL select instruction allows the application to create a filter to remove unnecessary information. For instance, in case that a mobile mashup application wants to obtain only the names and nicknames of a user's friends who tweeted most recently on Twitter, it sends the SQL select instruction
  SELECT name, nickname FROM https://api.twitter.com/1/statuses/home_timeline.xml?include_entities=true *WHERE* *.
  to the proxy to invoke the Web API
  https://api.twitter.com/1/statuses/home_timeline.xml?include_entities=true.
  When receiving the AQL select instruction, the proxy will extract only the desired names and nicknames after acquiring the response from the Twitter Web API server. Figure 7 shows the concise result for the requesting application after the proxy filters undesirable information using the AQL select instruction. With the AQL select instruction, the problem of Observation 3 can be resolved by enabling a mobile mashup application to obtain only the interested

Table 2 Format of AQL

Action	AQL	HTTP	Query format
Create	Insert	POST	**INSERT INTO** [API].[Method] ([Parameter Key(s)]) **VALUES** ([Parameter Value(s)])
Read	Select	GET	**SELECT** [Field]//[Attribute] **FROM** [API].[Method] **WHERE A** or **B A**: [Key Value Parameters] **B**: [Key] **IN** ([SubSelect])
Update	Update	PUT	**UPDATE** [API].[Method] **SET** [New Value] **WHERE** [Key Value Parameters]
Delete	Delete	DELETE	**DELETE FROM** [API].[Method] **WHERE** [Key Value Parameters]

```
<user>
  <name>Chenjm</name>
  <screen_name>Chenjm</screen_name>
</user>
```

Fig. 7 The result of using the AQL select instruction to extract the desired information

information without unnecessary information.

Moreover, the extra HTTP requests and responses caused by parameter dependence of Web APIs (mentioned in Observation 2) can be eliminated for mobile mashup applications through the use of an advanced AQL select instruction. As depicted in Fig. 8, when a mobile mashup application would like to know the districts of Taipei by Yahoo! Lifestyle API methods, the application uses the advanced AQL select instruction

SELECT FROM http://tw.Lifestyle.yahooapis.com/v0.3/Addr.listDistrict?
WHERE city= in
(SELECT ID FROM http://tw.Lifestyle.yahooapis.com/v0.3/Addr.listCity? *WHERE name=Taipei)*.

Upon receipt of such an instruction, the proxy first invokes the Web API http://tw.Lifestyle.yahooapis.com/v0.3/Addr.listCity? to retrieve the IDs of cities in Taiwan. Next, after receiving the returned IDs, the proxy invokes the following Web API with Taipei's ID *A*: http://tw.Lifestyle.yahooapis.com/v0.3/Addr.listDistrict?city=A.

After receiving the districts of Taipei, the proxy sends an HTTP response to inform the requesting application of its desired result. With the advanced AQL select instruction, the application is able to obtain the desired information with only one HTTP request and one HTTP response.

- **Update:**
  To update a resource on the Web API server, a mobile mashup application composes an AQL update instruction because a resource update is provided via an HTTP PUT method. After receiving an AQL update instruction, the proxy reverts to the original Web API and invokes the Web API for the application. For example, if a mobile mashup application wants to update one image in a Picasa Web album, the application sends the proxy the AQL update instruction
  UPDATE https://picasaweb.google.com/data/media/api/user/userID/albumid/ *SET photoID WHERE photoID* to ask the proxy to send the HTTP request
  PUT https://picasaweb.google.com/data/media/api/user/userID/albumid/albumID/photoid/photoID
  to the Picasa server. Similarly, the proxy invokes the corresponding Web API on behalf of the application and then sends the result from the Picasa Web API server back to the application.
- **Delete:**
  If a mobile mashup application wishes to delete certain resources on a Web API server, it composes an AQL delete instruction since an HTTP DELETE method is used for resource removal. Consider that a mobile mashup application wants to delete a specific Picasa album image. This application sends the proxy the AQL delete instruction
  DELETE FROM https://picasaweb.google.com/data/entry/api/user/userID/albumid/albumID *WHERE photoID*
  to ask the proxy to send the HTTP request
  DELETE https://picasaweb.google.com/data/entry/api/user/userID/albumid/albumID/photoid/photoID
  to the Picasa server. Similarly, the proxy calls the desired Web API for the application and then replies the requesting application with the response from the Picasa Web API server.

3.3 Image multi-get module

As mentioned in Observation 4, image retrieval APIs for Web albums or other type of Web sites that involve images (e.g., social networking sites) usually return a list of image URLs rather than the images themselves. As such, downloading n images of an album or from an online social page requires a total of $n + 1$ HTTP responses and $n + 1$ HTTP requests, incurring a substantial transfer cost for a mobile mashup application. Besides, the lack of options for quality, scale, and resolution of images may cause bandwidth waste and additional image processing time for applications. To address

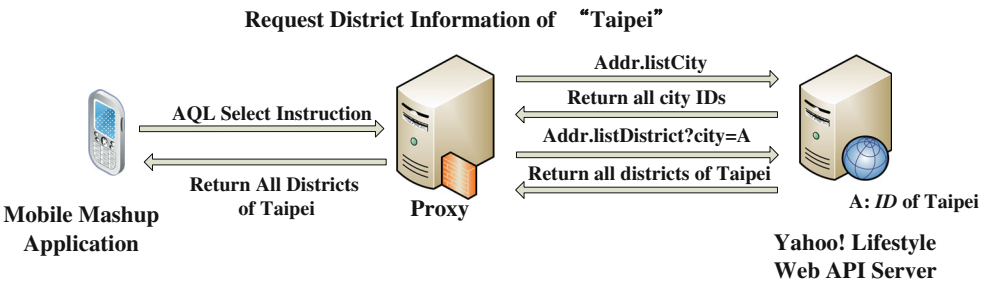


Fig. 8 AQL subselect instruction to address parameter dependence of Web APIs

these issues, we devise an *image multi-get (IMG) module* for mobile mashup applications. As illustrated in Fig. 9, the IMG module enables a mobile mashup application to download all images with only one HTTP request and response. When a mobile mashup application wants to download multiple images, it creates an IMG instruction that describes

- how to get the URLs of these images by Web API invocation, and
- how to adjust these images,

and sends the IMG instruction to the proxy. When receiving an IMG instruction, the proxy will request all images from the Web API server. Specifically, the proxy will send an HTTP request to the Web API server to retrieve the URLs of these images, and send n HTTP requests to the Web image server to retrieve the desired images. After obtaining all images, the proxy then proceeds to adjust these images according to the IMG instruction.

The format of an IMG instruction is "*SELECT [IMG].[Q].[S].[X].[Y] FROM [API].[Method] WHERE [Key Value Parameters]*". The options of an IMG instruction are quality (Q), scale (S), and resolution (X and Y), which are detailed below.

- **Quality ([Q]):**
  The parameter Q is used to enable a mobile mashup application to specify the desired quality of images (i.e., the degree of compression) for values of Q in the range of 1–100. If a mobile mashup application wants to obtain 80 % quality for the images returned from a Web image server, the application sets the IMG instruction to *SELECT [IMG].[80].[0].[0].[0] FROM [API].[Method] WHERE [Key Value Parameters]*.
- **Scale ([S]):**
  The parameter S is used to allow a mobile mashup application to shrink the received images for values of S in the range of 1–100. If a mobile mashup application would like to obtain images of 50 % scale for the images returned from a Web image server, the application sets the IMG instruction to *SELECT [IMG].[0].[50].[0].[0] FROM [API].[Method] WHERE [Key Value Parameters]*.
- **Resolution ([X].[Y]):**
  These two parameters X and Y are used to allow a mobile mashup application to specify the desired resolution (X pixels high by Y pixels wide) of images. If a mobile mashup application wants to receive images that are 60 pixels high by 90 pixels wide, the application sets the IMG instruction to *SELECT [IMG].[0].[0].[60].[90] FROM [API].[Method] WHERE [Key Value Parameters]*.

According to the received IMG instruction, the proxy adjusts the quality, scale, and resolution successively. Note that a parameter set to 0 or a value outside the defined range will be ignored by the proxy. The procedure of handling quality (parameter Q), scale (parameter S) and resolution (parameters X and Y) operations is as bellow.

- *Step 1:* If the value of the parameter S is not zero, the IMG module will resize the image with the same aspect ratio accordingly and then execute Step 3.
- *Step 2:* If none of the values of the parameters X and Y is zero, the IMG module will resize the image with width X pixels and height Y pixels.
- *Step 3:* The IMG module compresses the new image with the specified compression ratio.

After processing all images, the proxy will perform *image combination* by concatenating these images prior to transmitting the images to the requesting application. The motivation of image combination is to reduce the number of HTTP responses. To ensure that the requesting application realizes how to split the combined images, the proxy attaches the offsets of the concatenated images to the beginning of the payload of the HTTP response. Then, the images are concatenated at the end of the offsets (see Fig. 10). With the offset information, a mobile mashup application can obtain each separated image from the returned combined images while incurring lower packet overhead.

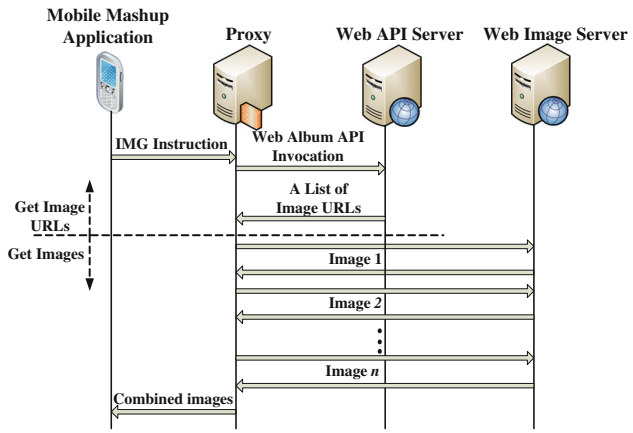


Fig. 9 Image Multi-Get Module

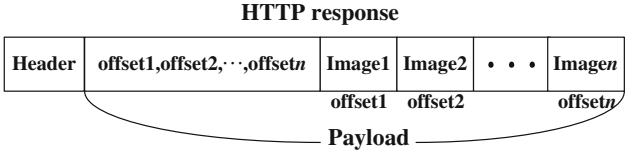


Fig. 10 HTTP response of the result of an IMG instruction

4 Performance evaluation

In this section, we conduct extensive experiments to evaluate the proposed proxy system and techniques. We implement a prototype of the proposed proxy on a desktop PC running Ubuntu 10.04.1 LTS with a P4 2.8 GHz processor and 4GB RAM. The experiments are conducted over a Wi-Fi network. To measure performance, we employ a wide variety of Web APIs and create several mobile mashup applications for an HTC Desire smartphone. The adopted performance metrics are listed below.

- Transfer size: We capture the HTTP request(s) sent from and HTTP response(s) received by the mobile mashup application using *tcpdump* installed on the HTC Desire smartphone. The HTTP request (labelled HReq) and response (labelled HRes) transfer sizes are used to validate the effectiveness of the proposed proxy system and techniques.
- Response time: Response time is defined as the elapsed time from the moment the mobile mashup application submits the first HTTP request to the moment all HTTP responses are received. To provide a better understanding of the benefit and overhead of the proposed system, we provide a breakdown of response time by dividing response time into the following three components:
  1. *Wireless transmission time (labelled WT)*: Wireless transmission time is the time spent sending the HTTP request(s) to and receiving HTTP response(s) from the proxy.
  2. *Proxy processing time (labelled PP)*: Proxy processing time is the time spent processing requests from the mobile mashup application and responses from the Web API server(s).
  3. *Resource access time (labelled RA)*: Resource access time is the time beginning from submission of HTTP request(s) to the Web API server(s) to the proxy receives all HTTP responses.
- Energy consumption: To evaluate the energy savings achieved by the proposed system and techniques, we use a hardware power monitor[2] to measure the energy consumption. The power monitor is attached to the battery of the HTC Desire smartphone and samples the current drawn from the battery at a frequency of 5,000 Hz. The energy measurement environment is shown in Fig. 11.

All experimental results are averaged over ten runs. The experimental results of the direct access approach are labelled "Direct", whereas those of the proposed system are labelled "AQL."

[2] Monsoon Power Monitor, http://www.msoon.com/.

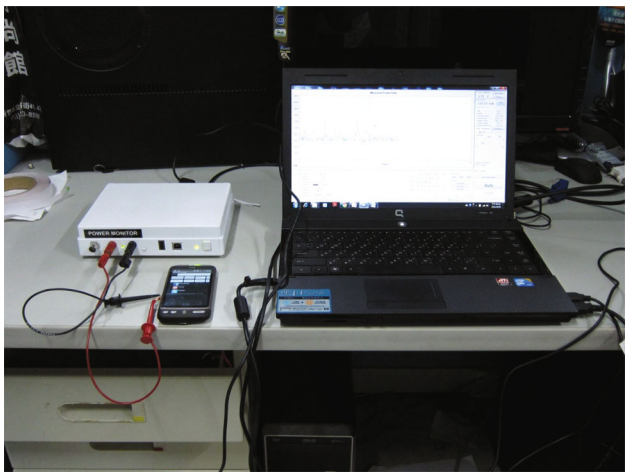


Fig. 11 Energy measurement environment

4.1 Experiment on unnecessary information filtering

Our first experiment investigates the effect of the proposed system on unnecessary information filtering (i.e., Observation 3). We select Web APIs from 10 top Web sites in the social, search, video and album categories based on the popularity of the Web sites. For social Web sites, we create multiple user accounts where each account subscribes to the social networking pages of celebrities and well-known companies (as listed in Table 3), and use Web APIs to retrieve their names, messages and URLs. For each search, video and album Web site, we use "Jeremy Lin" as the search keyword and the desired information as listed in Table 4.

Figure 12(a) shows that the proposed system reduces the total transfer size of the HTTP requests and responses by an average of 88.4 %. Specifically, the average reductions in transfer sizes of HTTP requests and responses achieved by the proposed system are 66.3 and 88.9 %, respectively. Regarding HTTP requests, the proposed system greatly reduces the transfer sizes for these Web APIs because the proxy handles some required parameters (e.g., the API key and the access token) for accessing Web resources that may be even larger than the parameters of some Web APIs. The transfer sizes of HTTP responses are significantly reduced due to the effective filtering of unnecessary information received from the Web API servers. Note that the result in Fig. 12(a) indicates that most Web API servers return a vast amount of unnecessary information to mashup applications.

Although the proposed system is able to significantly reduce the transfer size of Web API invocations, as depicted in Fig. 12(b), (c), the response time reduction and energy saving are only 9.2 and 22.1 %, respectively. As shown in Fig. 12(b), for the response time for the proposed system, wireless transmission time is a very

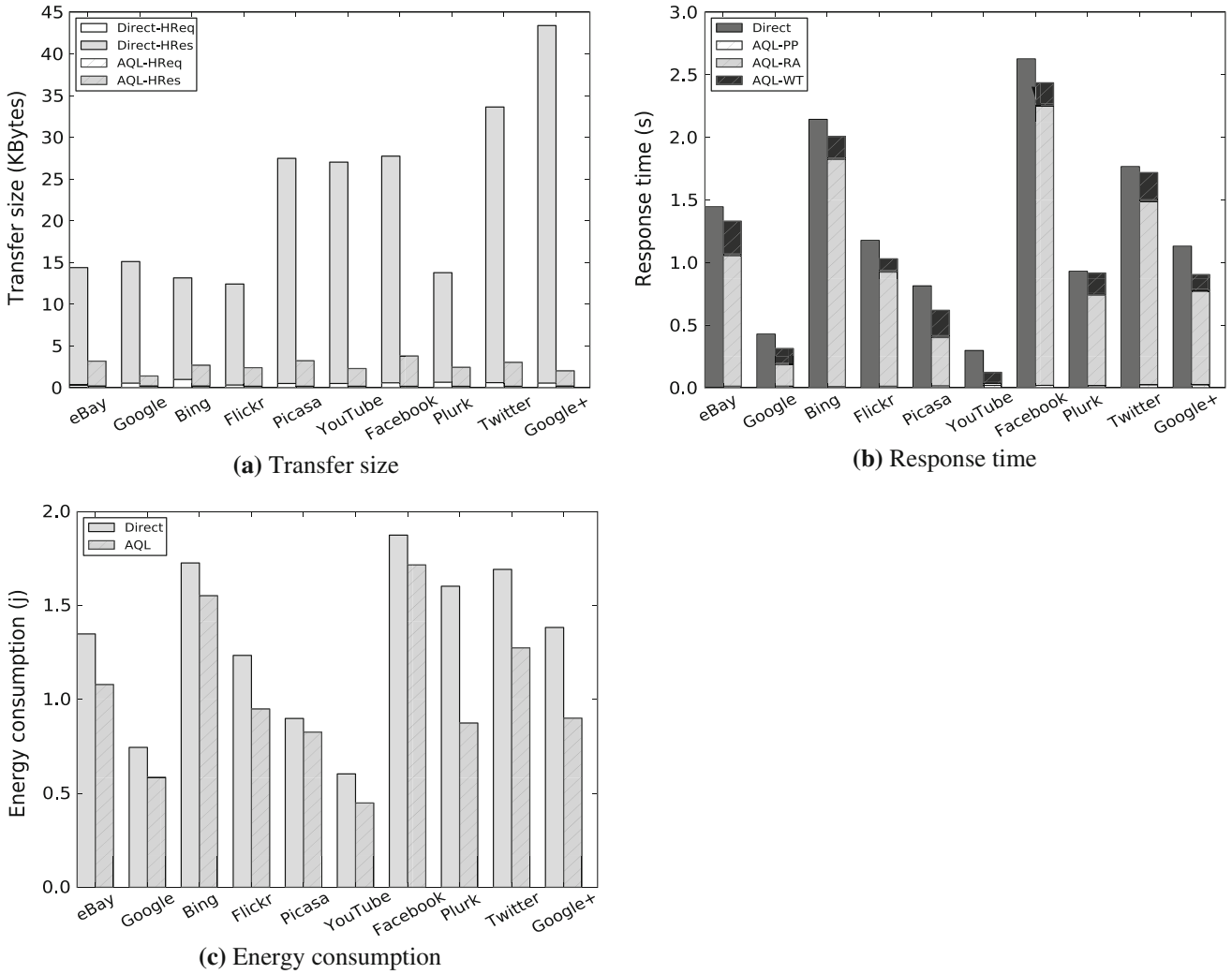
Table 3 List of subscribed social pages

Category	Social page
Celebrity	Jeremy Lin, Kobe Bryant, Jennifer Lopez, Carmelo Anthony, Lady Gaga, Bon Jovi, Michael Jordan, Christina Aguilera, John Nash
Company/ Service	CNN, Nike, Adidas, New York Times, Audi, Sony, HBO, ESPN, Starbucks, YouTube, iTunes Music

Table 4 List of desired information

Category	Service	Desired information
Album	Flickr, Picasa	Title, URL
Video	YouTube	Title, ID, Picture, URL
Search	eBay, Google, Bing	Title, URL
Social	Facebook, Plurk, Twitter, Google+	Name, Message, URL

small part while resource access time accounts for the majority. Because the proposed system is only able to reduce wireless transmission time, the performance gain of the proposed system regarding response time is limited. Figure 12(b) also indicates that proxy processing time is almost negligible and thus introducing the intermediate proxy does not result in significant overhead. Finally, because the energy consumption for accessing Web resources is mainly proportional to the active time



(a) Transfer size

(b) Response time

(c) Energy consumption

Fig. 12 Experimental results of unnecessary information filtering

of wireless interfaces, which is in proportion of the response time, the energy savings achieved by the proposed system is not as significant as reduction in transfer sizes.

4.2 Experiment on parameter dependency

This experiment studies the effect of AQL for resolving the problem of parameter dependency (i.e., Observation 2). We use Web APIs of YouTube, Facebook, and Picasa that require specific IDs prior to retrieving the desired information. We use "Jeremy Lin," "Starbucks" and "Olympics" as keywords to search the related videos, fan pages and albums. As shown in Fig. 13(a), the proposed system reduces total transfer sizes by 43.2–88.4 %. For HTTP requests, the proposed system reduces transfer sizes by 84 % on average. This is because the direct access approach requires two HTTP requests with large parameters whereas the AQL enables one HTTP request with lower parameter overhead. With respect to HTTP responses, the proposed system significantly improves performance, particularly for the YouTube and Facebook APIs. The reason for this is that those two APIs return considerable information in addition to the required IDs. Due to the reduction in HTTP responses, the proposed system achieves substantial reductions in transfer size that are similar to the results shown in Fig. 12(a). As explained previously, due to long resource access time, the significant reduction in transfer size of the proposed system results in relatively slight improvements in response time and energy consumption, as depicted in Fig. 13(b), (c), respectively.

4.3 Experiment on multi-image retrieval

In this experiment, we measure the effect of the IMG module on multi-image retrieval. We collect 30 images related to Olympics 2012 from Google Images, store them

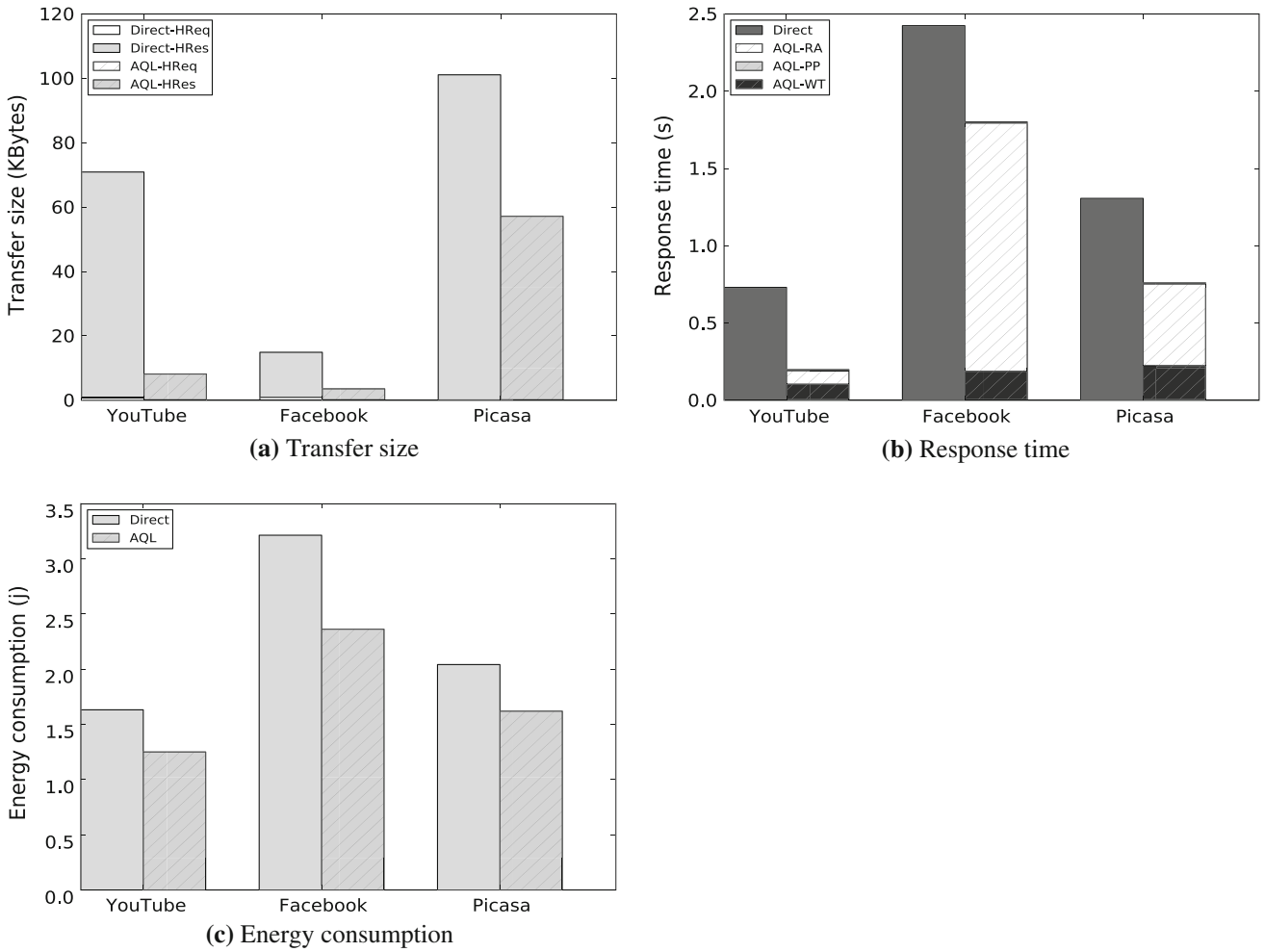


Fig. 13 Experimental results on parameter dependency

in Picasa, and adopt the Picasa Web APIs to retrieve the images. The number of the images ranges from 6 to 30 in increments of 6. To observe the effect of image adjustment, we also conduct the experiment with adjusted images. Parameterizations of the quality and the scale of requested images for AQL are denoted by *IMG:Q,S* where *Q* denotes the desired quality and *S* denotes the specified scale. For instance, *IMG:100,75* signifies that the application asks the proxy to return the images with 100 % quality and 75 % scale. We separately adjust the quality and the scale of the images; the average sizes of the adjusted images are presented in Table 5. Note that *IMG:100,100* does not adjust the images and is used to identify the benefit of image combination.

Figure 14 reveals that transfer size, response time and energy consumption all increase as the number of the images increases, as expected. From Fig. 14(a), it can be found that the transfer sizes of HTTP requests in the direct access approach are proportional of the number of images, whereas the transfer sizes of HTTP requests of the proposed system remain constant and are much smaller than those of the direct access approach. This finding is because the direct access approach requires $n + 1$ HTTP requests to download n images whereas only one HTTP request is submitted in the proposed system regardless of the requested number of images. Regarding the transfer sizes of HTTP responses, *IMG:100,100* reduces the transfer size by 23–34 % due to the benefit of image combination. Additionally, the transfer size of HTTP responses is significantly smaller when the images with degraded quality or small scales are accepted by the mobile mashup applications.

As depicted in Fig. 14(b), (c), it is interesting that, unlike the previous results, the proposed system achieves substantial reductions in response time and energy consumption. For response time, the proxy system shortens response time by 45.6–77 % and the performance gain is more significant with an increasing number of images. Due to the decrease in response time, energy consumption for the proposed system is as low as 23–48.8 % of the direct access approach, indicating the energy efficiency is achieved by the proposed system. Another interesting result is that *IMG:100,100* has shorter response times than *IMG:100,75* and *IMG:75,100* in spite of its larger transfer size, resulting from that *IMG:100,100* avoids the processing time for image adjustment (see Fig. 14(b)). Thus, the scheme using image combination only (i.e., *IMG:100,100*) is suitable for users that subscribe to unlimited data plans, whereas the schemes with image adjustment and image combination (e.g., *IMG:100,75*, *IMG:75,100*) are suitable for users that subscribe to usage-based pricing plans.

Table 5 Average sizes of the original and adjusted images

Original	230,799 bytes
IMG:100,100	230,799 bytes
IMG:75,100	49,751 bytes
IMG:100,75	146,676 bytes

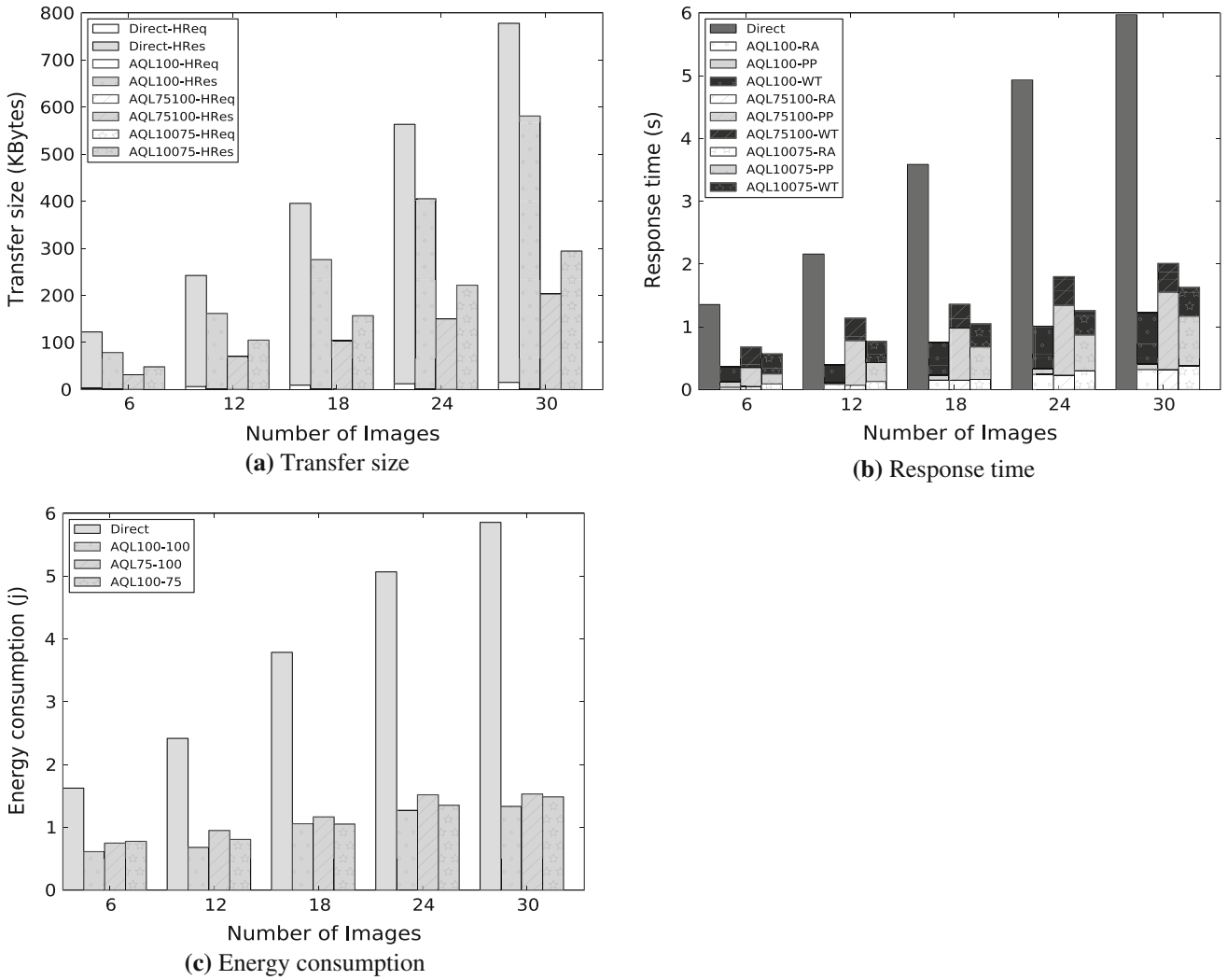
4.4 Experiment on the Friend Stream-like application

Finally, we implemented a Friend Stream-Like mashup application, which is shown in Fig. 15 to evaluate the overall benefit of the proposed system in this experiment. Our Friend Stream-like application separately fetches the most recently updated statuses of friends of mobile users from Facebook, Twitter, and Plurk. The number of fetched statuses varies from 5 to 20 in increments of 5. Note that we change neither the quality nor the scale of profile images (i.e., *IMG:100:100* is used) in this experiment. Figure 15 demonstrates that the proposed system reduces approximately 52.6 % of the total transfer size on average. Such a significant saving shows that the proposed system can effectively reduce the total transfer size for mobile mashup applications. For HTTP requests, the transfer size reductions achieved by the proposed system are approximately 73.5 % on average because only two HTTP requests (one Web API invocation for initialization and the other for retrieving profile image URLs) are submitted in the proposed system. Due to the increase in the number of image URLs, the transfer size of the proposed system becomes larger when the number of profile images increases. Due to the relatively slight improvement in the transfer size of images and the total image size that accounts for the majority of HTTP response sizes, the proposed system reduces the transfer size of HTTP responses by only 51.8 % on average. With smaller transfer sizes, the proposed system can reduce response time and energy consumption by 52.3 and 43.6 %, respectively, as plotted in Fig. 15(b), (c).

4.5 Remarks

From the above experimental results, we have the following two observations.

- AQL is able to greatly reduce the transfer size. However, the reductions in response time and energy consumption are not as significant as the reduction in transfer size.
  Basically, AQL is designed for text-based Web resource retrieval. Since text processing is usually of low complexity, the processing overhead resulting from the proposed proxy (i.e., proxy processing time) is negligible. The sizes of text-based Web resources are



(a) Transfer size

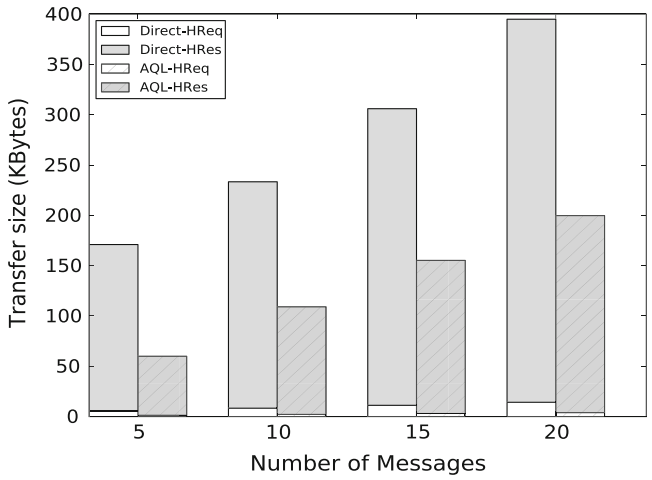
(b) Response time

(c) Energy consumption

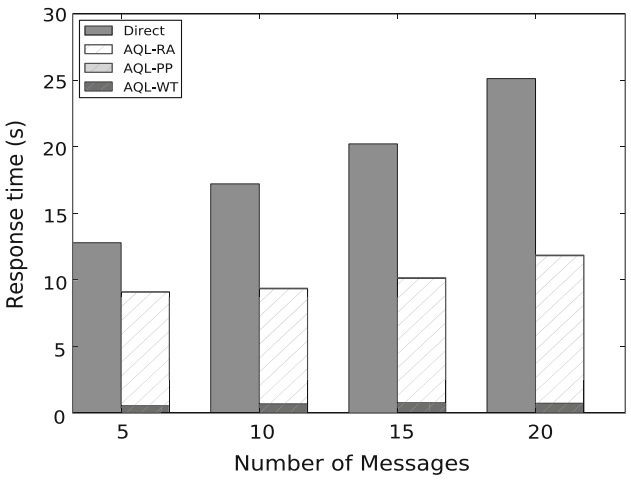
Fig. 14 Experimental results of multi-image retrieval

usually small (usually several kilobytes), thus, the wireless transmission time is much shorter than the resource access time. The resource access time is related to the capability of Web API servers, and cannot be reduced by proxy-side transfer size reduction. Therefore, the performance gain of the proposed system regarding response time is limited. Because the energy consumption for accessing Web resources is mainly proportional to the active time of wireless interfaces, which is in proportion of the response time, the energy savings achieved by the proposed proxy is not as substantial as the reduction in transfer sizes.

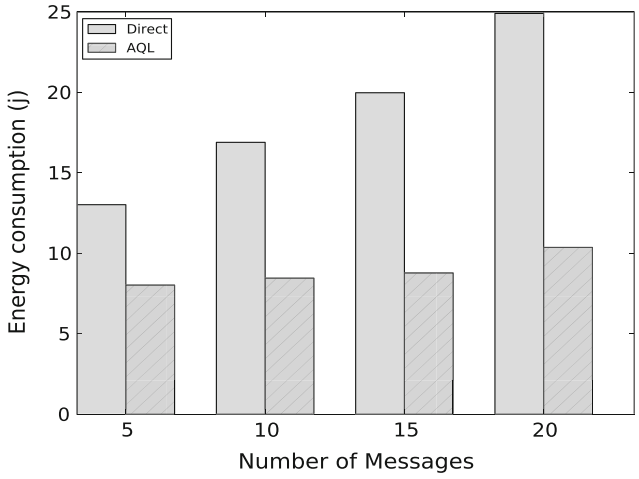
- Although the performance of IMG module in transfer size reduction is fair, the IMG module is able to greatly reduce the response time and energy consumption.
  Retrieving multiple images with only one HTTP request for the proposed system is particularly advantageous in a wireless environment, since the proxy is able to launch several threads to retrieve multiple images in a parallel manner. Therefore, the resource access time can be greatly reduced. In addition, with the IMG module, the reduction in energy consumption is also great since the wireless interfaces can quickly go to doze mode to save energy consumption due to short response time.
  Another interesting result is that although being able to achieve smaller transfer size, enabling image processing will incur longer response time and higher energy consumption. The reason is that enabling image processing will result in additional image processing overhead which is usually of high complexity. Therefore, the users subscribing to unlimited data plans are recommended to disable image processing for shorter response time and lower energy consumption, while the users subscribing to usage-based pricing plans are recommended to enable



(a) Transfer size



(b) Response time



(c) Energy consumption

Fig. 15 Experimental results for the Friend Stream-like application

image adjustment for lower transfer size as well as lower costs.

5 Conclusions and future work

In this paper, we proposed a proxy system and two techniques that enable energy-efficient and rapid Web API invocations for mobile mashup applications. We first studied the characteristics of Web APIs and made several observations about the cost of Web API invocations. Then, we proposed the use of an intermediary proxy combined with an API query language (AQL) and image multi-get (IMG) module to address the identified issues. The API query language allows mobile mashup applications to specify the information of interest and to address parameter dependency of APIs. On the other hand, the image multi-get module offers the adjustment of quality, scale, and resolution of images, and utilizes image combination to reduce HTTP header overhead. The proposed proxy system and techniques enable mobile mashup applications to receive the desired resources with fewer HTTP requests and responses as well as smaller HTTP responses. The extensive experimental results show that the proposed proxy system and techniques effectively mitigate the identified cost of Web API invocations and, thus, achieve significant improvement in transfer size, response time, and energy consumption for real applications.

Although the proposed approach is able to reduce response time and energy consumption by means of transfer size reduction, in order to take advantage of the proposed approach, a mobile mash application should be

implemented to use AQL to invoke Web APIs. In fact, the benefits of the proposed approach are expected to be attractive to mobile network operators (e.g., AT&T) and smartphone makers (e.g., HTC and Samsung). In recent years, some major smartphone makers implement some featured mobile mashup applications (e.g., HTC's Friend Stream) to differentiate their smartphones from other smartphones. With the proposed approach, one mobile network operator can deploy several AQL-enabled proxies in its network and cooperate with a smartphone maker to design a dedicated smartphone with several AQL-enabled mobile mashup applications. Therefore, the mobile network operator's customers can use these AQL-enabled applications with the advantages of energy efficiency and short response time. With transfer size reduction, the mobile network operator is able to use the same wireless bandwidth to serve more customers. In future work, we will develop a set of tools to help developers to compose, execute and debug AQL and IMG instructions. We believe that these tools will be able to relieve the difficulties of adopting the proposed approach. In addition, with the increasing popularity of smartphones, the scalability is an important factor of deploying the proposed approach. Thus, we will revise the proposed architecture to be able to utilize multiple servers to handle massive requests. We will also design a method to make the proposed architecture be able to dynamically add or shutdown severs on cloud computing platforms according to the sever load for better scalability.

Acknowledgments This work was supported by the National Science Council of Taiwan, ROC, under contracts 99-2221-E-009-140-MY2, 99-2219-E-002-029 and 101-2221-E-009-133.

References

1. Amazon. Amazon Silk. http://amazonsilk.wordpress.com/.
2. Carroll, A., & Heiser, G. (2010). An analysis of power consumption in a smartphone. In *Proceedings of the USENIX conference on USENIX annual technical conference*.
3. Chava, S., Ennaji, R., Chen, J., & Subramanian, L. (2012). Cost-aware mobile web browsing. *IEEE Perversive Computing 11*(3), 34–42.
4. Dogar, F. R., Steenkiste, P., & Papagiannaki, K. (2010). Catnap: Exploiting high bandwidth wireless interfaces to save energy for mobile devices. In *Proceedings of the 8th ACM international conference on mobile systems, applications, and services*, pp. 107–122.
5. Falaki, H., Lymberopoulos, D., Mahajan, R., Kandula, S., & Estrin, D. (2010). A first look at traffic on smartphones. In *Proceedings of the 10th ACM international conference on internet measurement*, pp. 281–287.
6. Falaki, H., Mahajan, R., Kandula, D., Lymberopoulos, R., Govindan, & Estrin, D. (2010). Diversity in smartphone usage. In *Proceedings of the 8th ACM international conference on mobile systems, applications, and services*, pp. 179–194.
7. Google. SPDY: An experimental protocol for a faster web. http://www.chromium.org/spdy/spdy-whitepaper.
8. Han, H., Liu, Y., Shen, G., Zhang, Y., & Li, Q. (2012). DozyAP: Power-efficient Wi-Fi Tethering. In *Proceedings of the 10th international conference on mobile systems, applications, and services*, pp. 421–434.
9. Han, R., Bhagwat, P., Lamaire, R., Mummert, T., Perret, V., & Rubas, J. (1998). Dynamic adaptation in an image transcoding Proxy for mobile web browsing. *IEEE Personal Communications 5*(6), 8–17.
10. Housel, B. C., Samaras, G., & Lindquist, D. B. (1998). WebExpress: A client/intercept based system for optimizing web browsing in a wireless environment. *Mobile Networks and Applications, 3*(4), 419–431.
11. Hsiu, P. -C., Lin, C. -H., & Hsieh, C. -K. (2011). Dynamic backlight scaling optimization for mobile streaming applications. In *Proceedings of the 17th IEEE/ACM international symposium on low-power electronics and design*, pp. 309–314.
12. Huang, J., Xu, Q., Tiwana, B., Mao, Z. M., Zhang, M., & Bahl, P. (2010). Anatomizing application performance differences on smartphones. In *Proceedings of the 8th ACM international conference on mobile systems, applications, and services*, pp. 165–178.
13. Kohavi, R., Henne, R. M., & Sommerfield, D. (2007). Practical guide to controlled experiments on the web: Listen to your customers not to the Hippo. In *Proceedings of the 13th ACM international conference on knowledge discovery and data mining*, pp. 959–967.
14. NetMarketShare. http://www.netmarketshare.com/report.aspx?qprid=61&sample=37.
15. Opera. Opera Mini & Opera Mobile. http://www.opera.com/mobile/specs/.
16. Pathak, A., Hu, Y. C., & Zhang, M. (2012). Where is the energy spent inside my app? Fine grained energy accounting on smartphones with Eprof. In *Proceedings of the 7th ACM European conference on computer systems*, pp. 29–42.
17. Pathak, A., Jindal, A., Hu, Y. C., & Midkiff, S. P. (2012). What is keeping my phone awake? Characterizing and detecting no-sleep energy bugs in smartphone apps. In *Proceedings of the 10th ACM international conference on mobile systems, applications, and services*, pp. 267–280.
18. Qian, F., Quah, K. S., Huang, J., Erman, J., Gerber, A., Mao, Z., Sen, S., & Spatscheck, O. (2012). Web caching on smartphones: ideal vs. reality. In *Proceedings of the 10th international conference on mobile systems, applications, and services*, pp. 127–140.
19. Shye, A., Scholbrock, B., & Memik, G. Into the Wild: Studying real user activity patterns to guide power optimizations for mobile architectures. In *Proceedings of the 42nd IEEE/ACM international symposium on microarchitecture*, pp. 168–178, December.
20. Yu, J., Benatallah, B., Casati, F., & Daniel, F. (2008). Understanding mashup development. *IEEE Internet Computing, 12*(5), 44–52.
21. Zhao, B., Tak, B. C., & Cao, G. (2011). Reducing the delay and power consumption of web browsing on Smartphones in 3G networks. In *Proceedings of the IEEE international conference on distributed computing systems*, pp. 413–422.
22. Zhong, L., & Jha, N. K. (2005). Energy efficiency of handheld computer interfaces: Limits, characterization and practice. In *Proceedings of the 3rd ACM international conference on mobile systems, applications, and services*, pp. 247–260.

Author Biographies

Chen-Che Huang received the B.S., M.S. and Ph.D. degrees in Mathematics from National Central University, in Computer Science and Information Engineering from National Dong Hwa University and in Computer Science from National Chiao Tung University, Taiwan, in 2004, 2006 and 2012, respectively. His research interests include mobile computing and spatial query processing

Jiun-Long Huang received his B.S. and M.S. degrees in Computer Science and Information Engineering Department in National Chiao Tung University in 1997 and 1999, respectively, and his Ph.D. degree in Electrical Engineering Department in National Taiwan University in 2003. Currently, he is an associate professor in Computer Science Department in National Chiao Tung University. His research interests include: mobile computing, wireless networks and data mining

Chin-Liang Tsai received the B.S. and M.S. degrees in Department of Information Management in National Taiwan University of Science and Technology and in Industrial Technology R&D Master Program of Computer in National Chiao Tung University, Taiwan, in 2006 and 2010, respectively. His research interests include mobile computing and Web services

Guan-Zhong Wu received the B.S. and M.S. degrees in Department of Computer Science and Information Engineering in National Chi-Nan University and in Department of Computer Science in National Chiao Tung University, Taiwan, in 2009 and 2012, respectively. His research interests include mobile computing and Web services

Chia-Min Chen received the B.S. in Department of Computer Science in National Chiao Tung University, Taiwan in 2010, and he is studying M.S. degree in Institute of Network Engineering, National Chiao Tung University, Taiwan currently. His research interests include mobile computing and P2P networks“

Wang-Chien Lee is an Associate Professor of Computer Science and Engineering at the Pennsylvania State University, University Park. He received a Ph.D. degree in Computer and Information Science from the Ohio State University and spent five years as a member of the technical staff at Verizon/GTE Laboratories, Inc. Dr. Lee performs cross-area research in data management, pervasive/mobile computing, and networking, with a special interest in spatial, temporal and multi-dimensional aspects. His work involves development of various techniques (including accessing, routing, indexing, caching, aggregation, dissemination, query processing, mining, and knowledge discovering) for supporting location-based services, recommendation services, social networking services, and complex queries in a wide spectrum of networking and mobile computing environments (such as mobile networks, wireless sensor networks, peer-to-peer networks, and wireless broadcast systems). Meanwhile, he also works on information retrieval, social computing, security, and Big Data. He has published more than 200 technical papers on these topics. He serves on the Editorial Board of IEEE Transaction on Service Computing, co-founded the IEEE International Conference on Mobile Data Management, and recently served as the technical program co-chair of the IEEE 2012 International Conference on Distributed Computing Systems