Geo-location-aware Emulations for Performance Evaluation of Mobile Applications

Alberto García Estévez
Universidad de Alcalá de Henares, Spain
alberto.garciae@alu.uah.es

Niklas Carlsson
Linköping University, Sweden
niklas.carlsson@liu.se

Abstract—**This paper presents the design of a simple emulation framework for performance evaluation and testing of mobile applications. Our testbed combines production hardware and software to allow emulation of realistic and repeatable mobility scenarios, in which the mobile user can travel long distances, while being served by an application server. The framework allows (i) geo-location information, (ii) client network conditions such as bandwidth and loss rate, as well as (iii) the application workload to be emulated synchronously. To illustrate the power of the framework we also present the design, proof-of-concept implementation, and evaluation of a geo-smart scheduler for application updates in smartphones. This geo-smart scheduler reduces the average download time by using a network performance map to schedule the downloads when at places with relatively good conditions. Our trace-driven evaluation of the geo-smart scheduler, illustrates the workings of the emulation framework, and the potential of the geo-smart scheduler.**

I. Introduction

Increased Internet speeds and wireless connectivity promises mobile clients seamless access to a wide range of Internet-based on-demand services, content and information. With increasingly mobile users and an exponential growth in the number of users connected to mobile data networks, mobile developers are increasingly building location-aware applications and services that provide customized service and take into account the client's geographic location.

While these applications and services are powerful and add a new dimension to online services, they come with new unique challenges. In this paper, we address the problem of how to evaluate new mobile applications that take into account the user location. Ideally, such evaluation framework should allow repeatable experiments (needed for fair head-to-head comparisons of alternative protocol implementations) using the mobility patterns and network conditions seen by real users.

While experiments using real test subjects address the second property, large-scale test deployments are typically expensive and do not allow repeatable experiments. System designers are therefore often limited to simulations and mathematical models when evaluating these systems.

This paper makes two primary contributions. First, in Section II, we present the design of a simple emulation framework for performance evaluation and testing of mobile applications. The framework combines basic production hardware and software to allow emulation of realistic and repeatable mobility scenarios, in which the user can travel long distances, while being served by an application server. The framework allows (i) geo-location information, (ii) client network conditions such as bandwidth and loss rate, as well as (iii) the application workload to be emulated synchronously.

Second, in Section III, we present the design and proof-of-concept implementation of a geo-smart scheduler for application updates in smartphones. We implement a notification service based on Google Cloud Messaging (GCM), which allows notifications to be pushed to multiple Android devices in parallel. However, rather than immediately downloading the contents that the client is informed about, our geo-smart scheduler creates, maintains, and use network performance maps to determine opportune times to download the contents; trying to reduce client download times and energy usage.

Using the emulation framework we show (in Section IV) that our geo-smart scheduler can reduce the average download times in some scenarios. While our results for the geo-smart scheduler is preliminary, the emulation-based performance evaluation illustrates the power of the emulation framework, as well as the potential of the geo-smart scheduler.

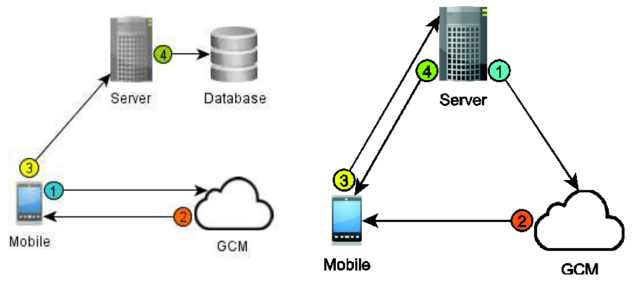
While previous works have demonstrated that the use of network performance maps can enhance the performance of some application in mobile data networks, these studies typically are based on simulations and/or theoretical analysis [1], [2]. To the best of our knowledge, and as discussed in Section V, this is the first paper to design, implement, and evaluate a real implementation within an emulation-based test environment. An extended version of the paper, as well as the code for our GCM-based server, geo-smart scheduler android app and emulator can be found at [3].

II. Emulation framework

A. Example application: GCM-based notification

To illustrate how our emulation framework can be used to compare the performance of alternative implementations, in this study, we use a basic notification service built using the Google Cloud Messaging (GCM) service.[1] GCM is free and can be used to push notifications to multiple Android devices in parallel. The service allows messages to be sent in both directions, and handles aspects such as queuing and delivery of messages to devices that are not online, for example. This type

[1]Google Inc., Developers Android - Google Cloud Messaging for Android, http://developer.android.com/google/gcm/index.html, 2013.



(a) Registration with GCM (b) Notification/download

Fig. 1: Overview of GCM-based example notification service.

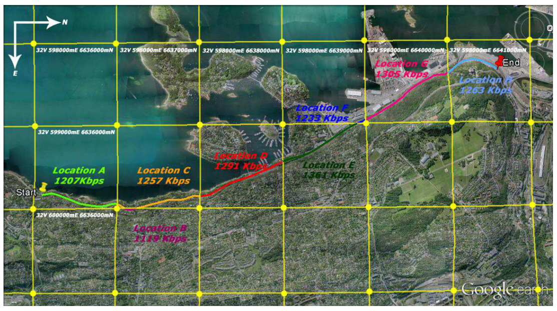


Fig. 2: Network performance map for the bus scenario (cf. Section IV-A) with 1000m resolution.

of implementation is very attractive in mobile environments with non-persistent connectivity.

The messages are encoded using JSON and are sent using either HTTP or XMPP. To register (Figure 1(a)) with the notification service: (1) the client sends a sender id and application id to the GCM server, (2) the GCM server register the client, and upon successful registration, issues a registration id to the android device, which (3) the client sends to the server, which then (4) stores the registration id for later use.

The server can now push notifications about objects of interest to the registered user(s) via GCM (steps 1 and 2 in Figure 1(b)). The client can later download these objects from the server (steps 3 and 4). In section III we describe a geo-smart scheduler that determines when these download requests (using HTTP) should be made, based on geo-location information and past history about network performance.

B. Trace-driven emulation testbed

Our emulation testbed consists of three main components.

- **Client running mobile application:** The notification application and geo-smart scheduler are both implemented and installed on a Wildfire powered with Android 2.3.7, root permission, Wi-Fi connectivity, and GPS.
- **GCM-based notification service:** The notification service is implemented using GCM, and pushes notifications to the mobile devices in order to inform them about new content available on the server.
- **Application server:** While the framework allows a generic server to be used, for the purpose of our experiments we use a laptop running an Apache server within Linux. PHP was used to develop the sever side application, and MySQL was used as a database.

Trace collection: To allow repeatable experiments, our evaluation framework relies on collection of realistic mobility traces that captures (i) the network conditions and coordinates seen by clients as they traverse the path, as well as (ii) the notifications generated by the application server. The first set of traces can be collected using example users, and should include a list of timestamp marked coordinates and network conditions as a client is moving through a network. The second set of traces can be collected at server side, and include the notifications and the timestamps when they were generated.

Trace-driven evaluation: The traces for the mobile's location and network conditions are feed into the mobile client while the notification traces are feed into the server, allowing us to capture the client mobility pattern, network conditions, as well as generating the notification workload.

Location mocking: To capture client mobility while the testbed remains in a simple lab environment, we have taken advantage of the built-in location providers (GPS and network provider) and the Android API to design code that mocks the mobile device to think that it is in motion along a route determined by the location trace. Our customized location provider is implemented in a java class, that registers a test provider on the device and sets the position of the location provider by periodically (every second) reading a pair of latitude and longitude coordinates from a file. Thereby, each time that the application needs to retrieve the location of the device, the location is requested from our test provider in the same way as with the GPS or network location provider.

Network conditions: To emulate the network conditions for each location along the client's recorded path we use dummynet [4] and a WiFi capable router that enable the connection between server and mobile. In our experiments dummynet runs within our server operating system and is configured to adapt the bandwidth conditions based on the location-bandwidth pairs read from a trace file.

In summary, combining the above components and their default hardware and software, we create an environment for testing mobile applications. By running the same network and workload traces for multiple candidate implementations, alternative solutions can be compared under the same conditions on the actual end-system devices.

III. Geo-smart scheduler

Our geo-smart scheduler is implemented using two complementing modules, which together aims to improve the resource usage of the mobile devices.

- **Map manager:** This module is responsible for creating and updating a network performance map, predict future performance seen by the client, and answer queries regarding network performance.
- **Download scheduler:** This module is responsible for determining when the client should schedule its downloads, such as to best leverage the available resources and predicted network performance seen by the client.

A. *Map manager*

Our network performance maps are stored in a database, and include tables with (i) throughput predictions for each visited location, and (ii) path predictions of upcoming locations. We implemented a map manager API (java library) that integrates all the functions and services needed to implement and manage these maps using three primary sub modules.

First, a *network monitor* passively collects sampled points of the cumulative HTTP byte counts and time stamps using the java `TrafficStats` class, as well as the Universal Transverse Mercator (UTM) coordinates using the `LocationManager` provided by the Android API. As illustrated in Figure 2, the UTM coordinate always corresponds to a square area where the side depends on the resolution of the coordinate and any point that lies within a square of a particular resolution has the same UTM coordinate value.

Second, all throughput-location pairs are sent to the *alignment module*, which takes these sample points and estimate the HTTP throughput as the bytes received between sample points divided by the time between sample points. An Exponential Weighted Moving Average (EWMA) is used to update the throughput estimate for each location. The alignment module is also responsible for updating the transition frequencies/probabilities between locations. Transitions with high frequencies indicate paths often taken by the user (e.g., a commuter). Combining the information of the current location with information for surrounding locations we can estimate the performance for longer time durations.

Finally, a *query interface* is provided that allow other modules, such as the download scheduler to query the network performance map about predicted/expected HTTP performance for the current location and along predicted future paths.

B. *Download scheduler*

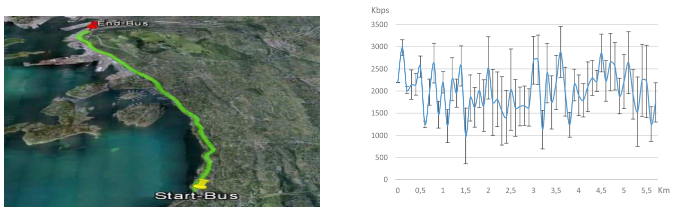
The current download scheduler uses a FIFO queue to handle notifications, and does not download objects out of order. When the device receives a notification about data available at the server, the scheduler uses the above query API to estimate the expected download performance and determine if the client should request the next file from the server or suspend download until reaching a better location.

To determine opportune times for downloads, we employ a simple threshold policy based on the expected average HTTP throughput, observed along the expected path over the duration of the download. When the expected average HTTP throughput exceeds the threshold or is unknown (i.e., the client is missing estimates for this location) a download is scheduled; otherwise, we wait until a zone change is detected, upon which a new estimate is requested from the map manager. While the threshold easily could be made adaptive, in our experiments, we use the average throughput seen by the client.

IV. Trace-based emulation evaluation

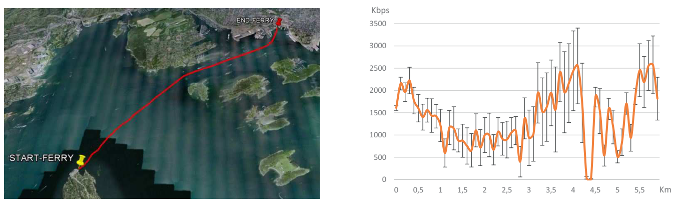
A. *Mobility scenarios*

While our framework allow arbitrary location and bandwidth traces to be used, the traces used for our performance



(a) Commuter path (b) B/w variation

Fig. 3: Bus scenario (5.8 km over 9 min 40 sec).



(a) Commuter path (b) B/w variation

Fig. 4: Ferry scenario (5.9 km over 9 min 52 sec).

evaluation were collected by a research group in Norway and captures two commuter scenarios [5], in which the commuters traveling into Oslo using bus (Figure 3) and ferry (Figure 4).[2]

In the bus scenario the average bandwidth (Figure 3(b)) vary greatly, but the minimum bandwidth of approximately 1.5 Mbit/s normally should provide the client with good download speeds. In the ferry scenario, the available bandwidth depends more strongly on the position along the path. The signal is strongest when the ferry is close to land, with an average bandwidth of 1.5 Mbit/s. In contrast, the signal far from land are weak. Although the signal is never completely gone while crossing the Oslofjord, the user rarely experience bandwidths greater than 1 Mbit/s in this segment of the path.

The location traces (sequence of time-location pairs) are feed to the mock location API in our emulation framework, making the mobile device believe it is in that particular location. The bandwidth traces (sequence of location-bandwidth pairs) are feed to the network emulator causing the client to see the bandwidth suggested by the trace.

B. *Notification workload*

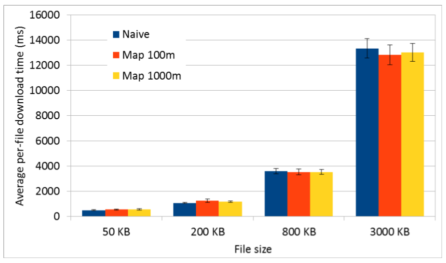
Notification traces were generated by collecting tweets with specific hashtags using the Twitter API.[3] The collection was performed for 3-4 hours for hashtags with high, medium, and low popularity, such as to allow traces with different notification frequencies. Each notifications trace include a list of unique tweet ids and their timestamps. The tweet id is used to keep track of which object the clients will try to download.

C. *Evaluation*

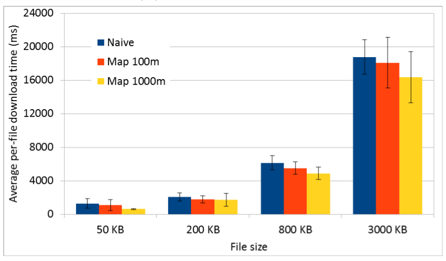
Our evaluation framework allow different policies to be compared under the same mobility scenarios and notification patterns. For illustration, the performance of the geo-smart scheduler have been evaluated using network performance maps with both 100m and 1000m location resolution, as well

[2]As there only is a limited number of traces available in the Riiser datasets [5], we also generated synthetic traces based on the original traces [3].

[3]Twitter, REST API v1.1, https://dev.twitter.com/docs/api/1.1, 2013.



(a) Bus scenario



(b) Ferry scenario

Fig. 5: Download times (measured in milliseconds).

as with a naive scheduler, which tries to download the content from the server as soon as a notification is received.

Figures 5(a) and 5(b) show the average file download time (per file) for different files sizes, for each of the policies, under the bus and ferry scenario, respectively. We note that there are between 10-20% reductions for the ferry scenario when using high granularity, but considerably smaller improvements for the bus scenario. The bigger savings in the ferry scenario are due to the scheduler avoiding to download during the times when the ferry is far from shore and performance is poor.

The relatively small improvements, especially for the bus scenario, are due to high variations in the network conditions, making the predicted values very unreliable. For example, we observe a normalized Root Mean Square Error (RMSE) between 0.3 and 0.5 in most locations, and high correlation (0.89 Pearson correlation coefficient) between the normalized RMSE and the coefficient of variation.

We note that part of the improvements with the 1000m granularity may come from the fact that there are more measurements for each location. Mobility can also play a factor in some cases. This is interesting as the biggest improvements in the ferry scenario is achieved for the 50KB case (50% reduction) for which transition between locations are unlikely.

While these results show that there are some potential for using geo-smart schedulers, more importantly, the case-based study clearly illustrates how our emulation framework can provide head-to-head performance comparisons of alternative implementations under relatively realistic scenarios in a simple and inexpensive lab environment.

V. Related work

Dummynet [4] and other network emulators [6]–[8] have been used to evaluate real systems under a wide range of scenarios. Different network performance maps [2], [9], [10] have been shown useful, including to help avoid TCP retransmissions [9] and to opportunistically schedule traffic across different provider links [10]. Perhaps most closely to ours is the work by Riiser at al. [11], who use the same traces as used in this paper to simulate the performance of a system that combines a reactive buffer-based HTTP-based Adaptive Streaming (HAS) algorithm with a simple prediction model to determine which video quality should be requested as a commuter watching video moves through the network. In contrast to these works, we present a relatively simple dummynet-based [4] evaluation framework that allow us to mock both the clients location and network conditions, and evaluate the advantage of a geo-smart scheduler that schedule downloads of (single quality) contents at opportune times, under a scenario in which the client has a single provider.

VI. Conclusions

This paper presents a simple approach for geo-location-based emulation and mobile application evaluation that allow repeatable experiments on real hardware, as well as the design, implementation, and evaluation of a geo-smart scheduler. The geo-smart scheduler implements a notification service that maintains network performance maps to determine opportune times to download contents from the applications servers.

Our emulation environment offers an effective way of testing applications in realistic mobility scenarios with minimum effort. Since the emulation environment uses real hardware, we can use all the features offered by mobile devices, including simple and fast software updates. By reducing the burden of the developer, while providing a framework for easy head-to-head protocol comparisons, this simple framework can greatly help designers improve their mobile applications. Future work include the design of adaptive policies and multi-granularity maps that make use of more advanced prediction algorithms.

References

[1] J. Yao, S. S. Kanheren, and M. Hassan, “An empirical study of bandwidth predictability in mobile computing,” in *Proc. ACM WiNTECH*, Sept. 2008.

[2] J. Yao, S. S. Kanhere, and M. Hassan, “Improving qos in high-speed mobility using bandwidth maps,” *IEEE Transactions on Mobile Computing*, vol. 11, no. 4, 2012.

[3] A. G. Estévez and N. Carlsson, “Geo-location-aware emulations for performance evaluation of mobile applications,” 2014. [Online]. Available: http://www.ida.liu.se/~nikca/papers/wons14.html

[4] M. Carbone and L. Rizzo, “Dummynet revisited,” *ACM SIGCOMM Computer Communication Review (CCR)*, Mar. 2010.

[5] H. Riiser, P. Vigmostad, C. Griwodz, and P. Halvorsen, “Commute path bandwidth traces from 3g networks: Analysis and applications,” in *Proc. ACM MMSys*, Feb/Mar. 2013.

[6] J. Ahrenholz, “Comparison of core network emulation platforms,” in *Proc. IEEE MILCOM*, Oct/Nov. 2010.

[7] E. Hernandez and A. S. Helal, “Ramon: Rapid-mobility network emulator,” in *Proc. IEEE LCN*, Nov. 2002.

[8] I. Ku, J.-T. Weng, E. Giordano, G. Pau, and M. Gerla, “Running consistent, parallel experiments in vehicular environment,” in *Proc. WONS*, Jan. 2011.

[9] K. Hojgaard-Hansen, T. K. Madsen, and H.-P. Schwefel, “Reducing communication overhead by scheduling tcp transfers on mobile devices using wireless network performance maps,” in *Proc. EW*, Apr. 2012.

[10] J. Yao, S. S. Kanhere, and M. Hassan, “Geo-intelligent traffic scheduling for multi-homed on-board networks,” in *Proc. MobiArch*, June 2009.

[11] H. Riiser, T. Endestad, P. Vigmostad, C. Griwodz, and P. Halvorsen, “Video streaming using a location-based bandwidth lookup service for bitrate planning,” *ACM TOMCCAP*, vol. 8, no. 3, July 2012.