CrowdMeter: An Emulation Platform For Performance Evaluation Of Crowd-Sensing Applications

Manoj R. Rege
Telecommunication Networks Group
Technische Universität Berlin.
Berlin, Germany
rege@tkn.tu-berlin.de

Vlado Handziski
Telecommunication Networks Group
Technische Universität Berlin.
Berlin, Germany
handzisk@tkn.tu-berlin.de

Adam Wolisz
Telecommunication Networks Group
Technische Universität Berlin.
Berlin, Germany
wolisz@tkn.tu-berlin.de

UbiComp'13 Adjunct, September 8–12, 2013, Zurich, Switzerland.

10.1145/2494091.2499578

Abstract
In this paper we introduce CrowdMeter, an emulation platform for predicting performance of large-scale crowd-sensing applications. CrowdMeter architecture follows natural decomposition of the application under evaluation and provides features for emulation of mobile devices and access network links. It leverages virtualization and cloud-based infrastructure-as-service resources to offer necessary scaling. We instantiate CrowdMeter architecture using off-the-shelf components and public-cloud resources, and perform a preliminary evaluation of its emulation fidelity focused on the communication services. The results confirm that CrowdMeter can successfully capture important aspects of real-world performance of different wireless access links. They also illustrate the ease-of-use and the scalability of the platform in terms of number of emulated mobile devices.

Author Keywords
Crowd-sensing, Emulation, Cloud, Virtualization

ACM Classification Keywords
C.4 [Performance Of Systems]: Measurement techniques

General Terms
Design, Experimentation, Human Factors, Performance

Introduction

The multitude of sensors available on the modern mobile devices have propelled *crowd-sensing* as promising approach for collecting valuable information about our physical environment, serving as enabler for many innovative applications in diverse domains [4] like environmental monitoring, health care, social networking, transportation, etc.

The rise of crowd-sensing applications can be primarily attributed to advancement of mobile devices as multi-utility platforms, easy and low cost availability of cloud technology, and ubiquitous network connectivity. The majority of existing crowd-sensing applications, typically organize the technology stack in three distinct tiers: a device tier that encompasses the application components that run locally on the user's mobile devices, a cloud tier that encompasses the back-end services, and an interconnection tier that links them. This technological base is complex and characterized by high level of heterogeneity. In an attempt to address this problem, several platforms have been recently proposed that offer customized programming abstractions for developing and deploying crowd-sensing applications [3, 10, 11]. They focus on hiding the underlying heterogeneity and on providing support for high-level specification of common functions such as description, distribution and scheduling of crowd-sensing tasks, and management of participation incentives.

Such support for developing the desired functionality is, however, not paralleled by any support for prediction of the performance of the developed crowd-sensing application. Various micro-level design decisions related to on-device data sensing, processing and storage, their offloading to cloud-based services, type and quality of the access links, user mobility, and behavioral dynamics can have profound and complex effect on performance of the application. As a result, the true performance is "just experienced" only after full deployment, when costs for taking corrective actions are very high.

The prevailing approach for predicting performance of crowd-sensing applications today involves recruiting volunteers and performing pre-deployment tests runs. While such pilot tests with volunteers are valuable for validating the basic functional correctness of the application, they provide only a first, simplistic information about performance as - by definition - they ignore the scaling factor and usually do not reflect the whole diversity of usage scenarios. Alternatively, a developer can pursue a purely simulation-based approach by leveraging the wide availability of existing models for different aspects like networking, mobility, etc. Unfortunately, available simulation models rarely adequately capture the relevant performance details of the mobile devices, communication between the devices and the cloud as well as the in-cloud data processing. This can lead to overall inaccurate performance estimates.

To address the limitations of both small scale prototyping and pure simulation, we present in this work *CrowdMeter*, an emulation-based platform for evaluating performance of large-scale crowd-sensing applications. Our platform leverages cloud-based infrastructure-as-service resources to support easy and realistic emulation of large numbers of participating mobile devices and their wireless access links, thus offering a valuable tool for assessing real-world performance impact of different design decisions even in pre-deployment development phase.

Besides CrowdMeter architecture, a core contribution of our work is the analysis of combined impact of mobile

device emulation and link emulation and virtualization. While each of these elements has been thoroughly investigated in isolation in the past, we believe that our work is the first to offer experimental data about the effects resulting from their interaction in the context of performance evaluation of large-scale crowd-sensing applications.

Design and System Architecture

The design of CrowdMeter platform focuses on three core tenets: emulation of different mobile devices, integration of human factors involved in crowd-sensing, emulation of different wireless access links as well as the computational effort needed to perform the processing in the cloud. We use a cloud based platform to emulate all the above components in a way which assures the possibility of their replication on large scale. In the following, we briefly explain some of the core challenges in each of these areas, and the design decisions that we took in CrowdMeter.

Emulating Mobile Devices

The realistic emulation of the participating mobile devices is a core element of CrowdMeter. The emulation provides software-based "imitation" of the major hardware components such as sensors, radios, CPU, memory, cache, etc. that play an important role in determining the crowd-sensing application performance. In this way it provides a virtual hardware substrate on top of which different software components can be instantiated and tested.

The quality of mobile device emulation has crucial impact on the capability of CrowdMeter to predict the overall performance of the application under evaluation. To this end we advocate using high-fidelity emulators based on mobile device hardware virtualization that offer portability and allow seamless migration of unmodified binary code between the real devices and their emulated doubles. Although these emulators offer an execution environment and functionality that closely mimics the modeled hardware, they might introduce subtle distortions and performance penalties, making it absolutely necessary to validate their behavior against real mobile devices. The validation step allows for quantifiable assessment of the possible impact that mobile device emulation might have on the predicted performance of an overall crowd-sensing application.

Modeling Human Factors

The mobile device provides the user the means to interact with the crowd-sensing application. The performance of such application is highly dependent on user behavioral characteristics and patterns. Factors such as user activity and context, mobility, user interaction patterns with the mobile device, and remaining battery capacity influence the likelihood of active user participation. The performance variations caused by these factors are likely to multiply on large scale and impact the overall application performance related to sensing coverage, sensed data quality associated with location accuracy, and sensor noise. In order to reliably predict application performance, it is necessary to model and introduce support for these human factors in the platform.

Emulating Communication Links

The wireless access technology and its characteristics have direct impact on the overall application performance, consequently the realistic emulation of communication links between the participating mobile devices and the cloud-based crowd-sensing application server(s) is another core element of CrowdMeter.

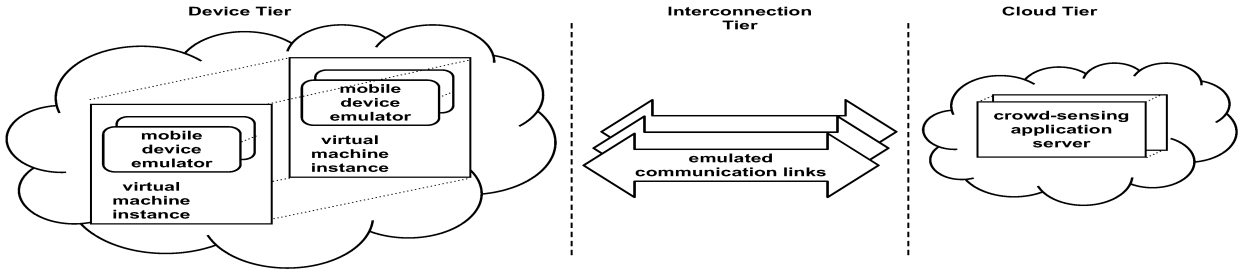


Figure 1: High-level architecture of the CrowdMeter platform

To maximize realism of emulation, interconnecting links in CrowdMeter are modeled on top of the real cloud-based interconnection network that is used by application under evaluation. To the best of our knowledge, the performance of communication link emulation on top of cloud based networks has not been studied in sufficient depth in the literature. Important design decisions associated with this task are related to the selection of the level of realism in emulation of individual links, and the location in the technology stack where this emulation should be implemented.

CrowdMeter Architecture

Crowd-sensing applications, typically organize the technology stack in three distinct tiers: a device tier that encompasses the application components that run locally on the user's mobile devices, a cloud tier that encompasses the back-end services, and an interconnection tier that links them.

The architecture of CrowdMeter closely mirrors this natural decomposition into three distinct tiers. Figure 1 depicts the high-level three tiered architecture of the proposed emulation platform.

The leftmost tier represents the emulated mobile devices (as discussed above) on top of a public cloud platform. It consists of a pool of emulator images from several heterogeneous mobile device platforms. A developer can create a population of heterogeneous emulated mobile devices at desired scale by selecting emulator images available in the pool. Each emulator image has device tier of the crowd-sensing application deployed on it.

The emulators are deployed on virtual machine instances of the underlying cloud platform. A virtual machine instance can have several multiple emulators deployed on it. Several factors such as level of realism, scaling latencies, monetary cost, etc. influence the decision of the number of deployed emulators per single virtual machine. The desired scale of the device population is achieved either by deploying more emulators on each virtual machine, or by instantiating new virtual machines from a template image containing the emulator and its configuration.

The rightmost tier is formed by the back-end servers, that are deployed in the hosting cloud. Unlike the previous tier, this tier is not emulated, however, it represents the real cloud tier deployment of the crowd-sensing application

under evaluation. In practice, above two tiers could also be deployed on the same public cloud platform.

Finally, the middle interconnecting tier in the architecture is comprised by the emulated links between the mobile devices and the back-end servers. For each mobile device replica, developers can select among a set of different wireless access link technologies that will be emulated on top of the wired communication infrastructure in the hosting cloud using standard network link emulators.

The use of virtualization in the cloud can introduce unwanted artifacts in the networking communication links [14, 15]. Common problems include long-tailed latencies, and unstable throughputs. Interestingly, these artifacts are typically not result of the public cloud network infrastructure but rather effects introduced by processor sharing on virtualized machines on the end network link points. Furthermore, they are also dependent on the amount and type of load sharing on the underlying CPU resources.

In case of CrowdMeter, these artifacts can be even more severe and pronounced due to presence of mobile device emulation on virtualized environment. This can lead to reduced fidelity in the emulated links thereby influencing the overall application behavior. For a successful prediction of the overall performance of the crowd-sensing application under evaluation, the link emulation support in CrowdMeter has to be able to produce link characteristics that are similar to the real access network links, despite the combination of several artifacts from emulation and virtualization.

Modeling Large-Scale Systems

The high fidelity of the emulation of the individual mobile devices and interconnection links would mean little without support for their efficient replication, as necessary predisposition for realistic time, this efficient scaling should not come at the cost of reducing the emulation quality of the individual components.

To support the necessary scales, our CrowdMeter platform leverages infrastructure-as-service resources from the public cloud to deploy large number of instances of the emulated mobile devices and links. Since the resource sharing in the cloud can cause variations in emulation behavior by introducing biases such as additional latencies, it is important that these effects are identified and quantified in order to prevent them from influencing accuracy of the performance evaluation of the overall crowd-sensing application under evaluation.

To this end, it is necessary to benchmark different common crowd-sensing tasks such as sensing, local data processing and communication under the *joint effects of emulation and virtualization*. The authors in [9, 13] provide several benchmark tests for computation, networking, and storage on the public cloud platforms and highlight the performance gaps existing in the services offered by public cloud platforms for different categories of application.

CrowdMeter Instantiation

Having presented the generic CrowdMeter design and architecture, in this section we discuss the motivation behind the concrete instantiation of its components.

In instantiating the CrowdMeter architecture, we aimed at reuse of off-the-shelf and publicly available components as much as possible, allowing us to ascertain the quality of performance prediction that can be readily made available to a large number of crowd-sensing application developers within acceptable cost boundaries.

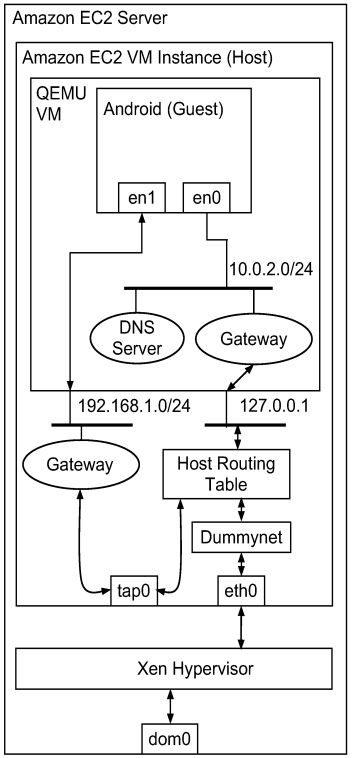


Figure 2: Network communication elements and flow pattern inside a single mobile device emulator in CrowdMeter - An Android (Guest) based mobile emulator in the QEMU virtual machine is deployed on a Amazon EC2 virtual machine instance (Host) running on top of Amazon EC2 Server using a Xen Hypervisor.

In this paper, we focus our discussion on two major components of emulated application: the mobile devices, and the access link. In addition we discuss the details of the cloud platform used for hosting the emulators.

Mobile Emulation Platform
The mobile device emulation in the current instance of CrowdMeter, is based on the Android emulation platform [1]. The platform supports the emulation of the entire Android stack including the underlying kernel. The Android emulator is implemented using the Quick EMUlator (QEMU) virtualization suite that supports emulation of several different CPU architectures on a host machine. QEMU uses dynamic translation for converting binary opcode of the emulated CPU architecture to the underlying host machine opcode.

In addition to the CPU emulation, QEMU provides networking support to the Android guest environment. The detailed network communication elements and flow pattern between the Android emulator and host are depicted in Figure 2. In user mode networking, the Android guest is connected to the underlying host network through a gateway. However, the guest is not directly accessible from host machine or an external network. Therefore, we use alternate QEMU networking mode by configuring an additional network interface on the Android guest (en1), using gateway interface (tap0) on the host, and IP forwarding to communicate with host default interface (eth0).

The emulator platform uses Android Virtual Device (AVD) that includes custom profiles for hardware. CrowdMeter uses AVD configuration encapsulating Android 2.3.3, ARM processor, 1 GB of RAM and SD-card storage, along with support for GPS, camera, accelerometer and audio sensors emulation.

Mobile communication link emulation
A natural choice to instantiate link emulation is on the emulator of the mobile devices themselves. The Android emulator indeed has support for different kinds of link emulation (using a simple leaky bucket implementation). Our in-depth studies, however, have shown that the link emulation functionality of the Android emulator is unstable and does not honor link bandwidth caps due to a design and implementation bug. These problems have been identified and reported back to the Android emulator developer community[1].

Due to these limitations and the fact that the Android emulator does not support link emulation on multiple network interfaces, we were forced to seek an alternative solution. After evaluation of several network link emulators, we have decided to perform the link emulation in CrowdMeter on the underlying host machine instance with the help of Dummynet [12].

In the current instance of CrowdMeter we support mobile access link emulation for 3G and WiFi networks. The access links are emulated by defining the bandwidth, delay and the packet loss parameters based on the communication technology. Table 1 gives the bandwidth and the delay parameters used by CrowdMeter for emulating 3G and WiFi networks, which have been adapted from published experimental measurements studies [6].

The Hosting Cloud Platform
From the plethora of available choices for the cloud hosting platform [9], we have selected Amazon EC2 [2] for the current instantiation of CrowdMeter architecture.

[1]Android Emulator Net Speed - Throughput not capped https://android-review.googlesource.com/#/c/61620/

Table 1: Mobile access link emulation parameters

Link Type	3G	Wifi
Delay (ms)	150	64
Uplink (Mbps)	1.01	0.93
Downlink (Mbps)	1.22	4.66

Amazon EC2 uses Xen virtualization technology and provides instance abstraction for each virtual machine that is run on the Amazon server (shown in figure 2). Each server has several running instances of the virtual machine. CrowdMeter allows usage of three different classes of virtual machine instances provided by Amazon EC2: Micro, Small and Medium as hosting platform for deploying the mobile device and network link emulators.

Table 2: Amazon EC2 instance types and specifications

Instance Type	Cores	Memory (GiB)	Network Performance
Micro	1	0.615	Very Low
Small	1	1.7	Low
Medium	2	3.75	Moderate

A more detailed specification of the individual type of EC2 instances is given in Table 2. CrowdMeter provisions all virtual machine instances using a shared template image with a Ubuntu 12.2 Linux distribution. The platform is deployed in the eu-west-1a availability region of Amazon EC2.

Preliminary Evaluation

In this section, we discuss methodology and tools for evaluating the CrowdMeter platform, and report on the initial results from the evaluation of its communication components. They enable us to build first impressions about the fidelity of the platform and its emulation capability of different mobile-to-cloud access technologies and protocols that play an important role in the overall performance of most crowd-sensing application.

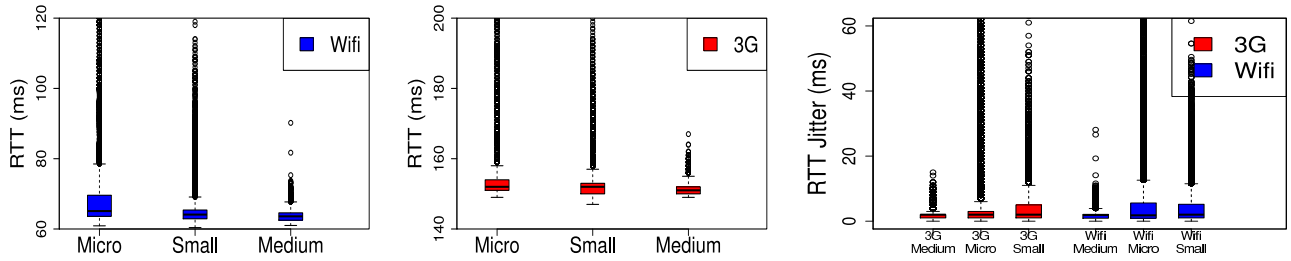
For all our evaluation experiments, we consider an EC2 Medium instance in the same zone as our crowd-sensing application server and the mobile emulator clients are deployed on Micro, Small and Medium instances respectively.

In the following we first provide a short overview of the performance metrics that we use to assess the quality of the emulation of the access link technologies and networking protocols.

Round Trip Time (RTT) - RTT metric offers insight into variations in communication latencies that largely influence behavior of network protocols used for interaction between mobile and cloud services. RTT is important metric for capturing performance of delay sensitive crowd-sensing applications.

Bandwidth – TCP Throughput - Bandwidth represents available network capacity variations over time from heterogeneous factors such as mobility of participating user(s), number of active network users, network load etc. TCP is commonly used transport protocol by crowd-sensing applications for communicating with cloud services. Transport throughput is important metric for capturing performance of crowd-sensing applications generating rich multimedia sensing data on mobile devices.

We use the standard UNIX Ping tool to measure RTTs. We collect 2000 RTT measurements by sending 10 packets per second. TCP throughputs on the emulated links are measured using IPerf [8]. For the purpose of TCP throughput measurements, we use sender and receiver window sizes of 256 KB respectively. All evaluation measurements are collected across 10 different pairs of emulated mobile access links between mobile device emulator running on a virtualized instance and a server deployed on a Medium instance. We measure TCP throughputs across 5 minute intervals in uplink and downlink direction with a resolution of 0.5 s. All



(a) Wifi (b) 3G (c) RTT Jitter

Figure 3: Boxplots of measured RTT, and RTT Jitter of emulated 3G and WiFi links between mobile device emulator client deployed on Micro, Small and Medium instances and the crowd-sensing application server deployed on a Medium instance.

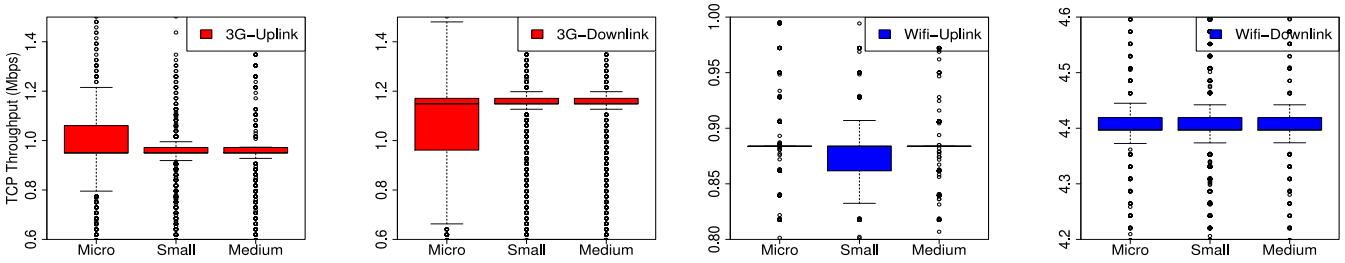
evaluation measurements are carried out sequentially in uplink and downlink direction respectively.

Round Trip Time (RTT)
Figure 3 shows the RTT and the RTT jitter boxplots of emulated mobile access links between the mobile device emulator client on virtualized instance and the server. The box plots denote 5th, 25th, 50th, 75th and 95th percentile of measurements for each type of link. The number of hops between clients and server were measured using the UNIX traceroute tool and vary from 4 to 7. Link emulation is applied by adding delay to incoming and outgoing ICMP packets (150 ms and 64 ms for 3G and WiFi links respectively).

In figure 3, the joint effect of emulation and virtualization is quite evident in longer tails in distribution of measured RTTs of emulated links. Previous studies [14] have shown longer tails on barebone virtual instances as a result of CPU sharing on the underlying server. We observe that Micro instances which give low network performance have the longest tail of all three followed by Small and Medium instances respectively. With respect to performance of delay introduced by link emulation, we see that Medium–Medium instance pair impose tighter bounds on RTTs with 99 percentile of RTTs within 155 ms and 67.8 ms for 3G and Wifi compared to 1019 ms and 988 ms for the Micro–Medium, and 177 ms and 89.6 ms for the Small-Medium pair respectively.

Table 3: Comparison between Average RTTs (ms) of emulated 3G and Wifi links measured directly on underlying host virtual instance and those on mobile emulator on a virtual instance for Micro, Small and Medium instance types.

Instance	3G		Wifi	
	Host	Emulator	Host	Emulator
Micro	154.7	182.6	64.7	94.5
Small	151.3	153.4	64.6	66.1
Medium	149.5	151.2	62.5	63.64



(a) 3G-Uplink (b) 3G-Downlink (c) Wifi-Uplink (d) Wifi-Downlink

Figure 4: Uplink and Downlink TCP throughputs of emulated 3G and Wifi links on Micro, Small and Medium instances respectively.

Table 3 highlights the difference in average RTTs measured on the virtualized emulator running on a virtual instance and on the same underlying host virtual instance. The increase in the RTTs are due to two factors, firstly, the network configuration of the emulator running on QEMU connected through a gateway with forwarding to the host virtual instance, and secondly, running QEMU emulation on a virtual instance. The amount of impact caused by virtualization can be seen by reduction in RTT difference in high-end instance types. In case of Micro instances, virtualization degrades the performance of the emulator to the point where average RTT on the emulator compared to the host virtual instance is higher by 27.9 ms (3G) and 29.7 ms (WiFi).

Next, we compare RTTs of the emulated links to real world crowd-sensed 3G and WiFi measurements reported across different mobile carriers in the USA by [6, 7]. RTTs for 3G networks reported from different carriers range from 140 ms to 340 ms while WiFi networks reported RTTs were typically anywhere between 50 ms to 400 ms. Median RTT jitter was observed to be more than 10 ms for 3G and around 7.9 ms for the WiFi links.

Comparing RTT ranges and the jitter values of the CrowdMeter emulated links to these statistics, we see that, although, Medium–Medium and Small–Medium pairs give tighter bounds in RTTs for emulated links for the defined parameter, they do not emulate wider range of RTTs and jitter in network links. RTT of emulated links is a characteristic of the link emulation algorithm used (in this case adding constant delay on each packet by Dummynet). We observe that simply by using the median values of the RTTs from real 3G and WiFi networks as delay parameters coupled with longer tails in the RTTs induced by virtualization, low-end Micro and Small instances reproduce RTT behavior that have characteristics similar to the RTTs of real 3G and WiFi network links.

TCP Throughput

Figure 4 summarizes TCP throughput measurements between each type of instance pair viz. Micro–Medium,

Small–Medium and Medium-Medium in uplink and downlink direction respectively. Again, we use median values of throughputs reported for 3G (1.01 Mbps and 1.22 Mbps) and WiFi (0.93 Mbps and 4.66 Mbps) as the limiting bandwidth for uplink and downlink emulation respectively. In figure 4, we observe that all three instances have same median TCP throughput 0.95 Mbps (3G-Uplink) and 0.88 Mbps (WiFi-Uplink), 1.14 Mbps(3G-Downlink), 4.39 Mbps (WiFi-Downlink). The average TCP throughputs are capped by bandwidth parameter specified for the link emulation. From these results, we see that average TCP throughput performance of Micro, Small and Medium instance in uplink and downlink are similar (given in Table 4).

Table 4: Average TCP Throughputs (Mbps) of emulated 3G and Wifi links on mobile emulator.

Instance	3G		Wifi	
	Uplink	Downlink	Uplink	Downlink
Micro	0.98	1.09	0.86	4.38
Small	0.94	1	0.87	4.35
Medium	0.95	1.14	0.87	4.39

Analysis of instantaneous TCP throughput plots show throughput periodically fluctuating between higher periods of data transfer followed by stable throughput period and a sudden drop in both uplink and downlink direction. With using link emulation, this behaviour is not as pronounced as observed in some of the earlier studies [14].

Having analyzed performance of link emulation at network and transport layers, supported by CrowdMeter platform, we learn that link behavioral characteristics of the emulated links on different virtual instances match closely to mobile access links in diverse mobile user context scenarios in reality. Our observations are based solely on the comparison of end statistics of the emulated links and real measurements in literature. We believe that in future this can be exploited in intelligently selecting instance type depending on crowd-sensing performance scenario to be evaluated.

Group Communication

Finally, we evaluate CrowdMeter platform for scalability and for modeling of large-scale group communication at the application level. In many diverse classes of crowd-sensing applications there is an absolute need for notifying a large group of mobile devices (or participants) for sensing the phenomenon of interest. Very often due to need of increased sensing data fidelity and coverage, a group of devices or participants need to be notified in a timely manner.

Many applications use Push based messaging protocols for enabling such notifications. We use Google Cloud Messaging (GCM) [5] protocol to evaluate the fidelity of links at application level under scalability. GCM enables cloud services (crowd-sensing cloud tier) to send messages to their respective Android based device applications (device tier) through a dedicated GCM server.

We want to demonstrate the use of CrowdMeter in analyzing the impact of scaling number of devices in a GCM multicast notification query on the best effort delivery latency. The latency is the application level latency and is the time between submitting a multicast notification request from crowd-sensing application server to GCM server and receiving a notification on the device.

Using CrowdMeter, we deploy a population of 200 virtual instances of Small type with Android emulators. Of the 200 deployed emulators, we observed that only one failed during our experiments. We create a mix of emulators in

the population using 3G and WiFi networks with the communication link emulation support. Further, we prototype and deploy on 200 emulators an Android application that uses GCM to receive message notifications from the crowd-sensing application server on a medium instance.

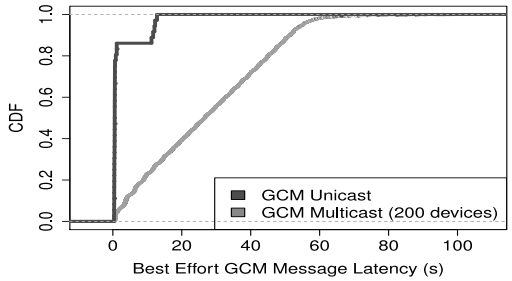


Figure 5: Impact of scale on the best effort GCM message delivery latency

Best effort delivery of a GCM message can be defined by specifying the time-to-live field of GCM message to 0, which ensures that a message is either delivered immediately or dropped. In order to measure notification message latency, we synchronize the emulators on each instance with the crowd-sensing server using NTP.

For our validation experiments, we push single notification every 30 s by setting the time-to-live to 0. We carry out 20 runs each time varying the percentage of 3G and WiFi devices in the crowd in increments of 10. Figure 5 shows CDF of the latencies measured on all mobile emulators using both 3G and WiFi link emulation. The CDF highlights the impact of scaling the number of devices on the best effort message delivery latency in a single multicast notification request. The increase in median best effort delivery latencies from unicast notification messages (0.54 s) to multicast notifications to 200 devices (26.5 s) is around order of magnitude of 50.

Conclusion and Future Work

In this paper, we presented CrowdMeter, a cloud-based emulation platform for carrying performance evaluation of crowd-sensing applications. Using a combined approach based on emulation and virtualization, CrowdMeter focuses on addressing the challenges in predicting the performance of large-scale crowd-sensing applications in the pre-deployment phase.

Our preliminary evaluation of the communication and networking components of the CrowdMeter platform, instantiated with an Android based mobile emulator and using the Amazon EC2 service, was concentrated on the assessment of the impact of the virtualization in the cloud environment on the fidelity of the mobile access link emulation, observed through delay and throughput metrics.

The presented results show that in terms of delay emulation, low-end and cheap cloud instances with their long tailed latencies offer link emulation performance that is most similar to real mobile access links. In terms of emulation of available bandwidth, the results show that the amount of virtualization does not impact the average TCP throughput in the emulated links. The instantaneous throughput, however, can be highly variable.

We also evaluated CrowdMeter for scalability using a case-study using the Google Cloud Messaging service which is commonly used in several crowd-sensing applications. The GCM results highlight the impact of scale on the message dissemination latencies and demonstrate the ease of use and the scalability of CrowdMeter.

We plan to build on the existing evaluation results and further improve the platform functionality. Targeting even larger scales would require more efficient scaling strategies. To this end we plan to investigate more closely the challenges associated with deploying multiple mobile emulator components on a single virtual instances in the host cloud platform. Further, modeling platform support for human behavioral factors associated with crowd-sensing is another aspect that we intend to focus on in future.

In the next period we will evaluate the emulation fidelity of additional CrowdMeter components for different common crowd-sensing application tasks like sensing, data processing, computation offloading, and data storage, as core building blocks of many crowd-sensing applications. We also intend to open the CrowdMeter platform and provide suitable programming abstractions and APIs that can simplify the specification of custom performance prediction scenarios for the developers.

Acknowledgements

The work has been partially supported by EIT ICT Labs, as part of the activity 12149, "From WSN Testbeds to CPS Testbeds".

References

[1] Android Emulator. http://developer.android.com/tools/help/emulator.html.

[2] Amazon Web Services. http://aws.amazon.com/.

[3] Das, T., Mohan, P., Padmanabhan, V. N., Ramjee, R., and Sharma, A. Prism: Platform For Remote Sensing Using Smartphones. In *Mobisys* (2010), 63–76.

[4] Ganti, R. K., Ye, F., and Lei, H. Mobile Crowdsensing: Current State And Future Challenges. *IEEE Communications Magazine 49*, 11 (2011), 32–39.

[5] Google Cloud Messaging For Android. http://developer.android.com/google/gcm/index.html.

[6] Huang, J., Qian, F., Gerber, A., Mao, Z. M., Sen, S., and Spatscheck, O. A Close Examination Of Performance And Power Characteristics Of 4G LTE Networks. In *MobiSys* (2012), 225–238.

[7] Huang, J., Xu, Q., Tiwana, B., Mao, Z. M., Zhang, M., and Bahl, P. Anatomizing Application Performance Differences On Smartphones. In *MobiSys* (2010), 165–178.

[8] Iperf. http://iperf.sourceforge.net/.

[9] Li, A., Yang, X., Kandula, S., and Zhang, M. Cloudcmp: Comparing Public Cloud Providers. In *IMC* (2010), 1–14.

[10] Ra, M.-R., Liu, B., La Porta, T. F., and Govindan, R. Medusa: A Programming Framework For Crowd-sensing Applications. In *MobiSys* (2012), 337–350.

[11] Ravindranath, L., Thiagarajan, A., Balakrishnan, H., and Madden, S. Code In The Air: Simplifying Sensing And Coordination Tasks On Smartphones. In *HotMobile* (2012), 4:1–4:6.

[12] Rizzo, L. Dummynet: A Simple Approach To The Evaluation Of Network Protocols. *ACM Computer Communication Review 27* (1997), 31–41.

[13] Walker, E. Benchmarking Amazon EC2 For High-performance Scientific Computing. *LOGIN 33*, 5 (Oct. 2008), 18–23.

[14] Wang, G., and Ng, T. S. E. The Impact Of Virtualization On Network Performance Of Amazon EC2 Data Center. In *INFOCOM* (2010), 1163–1171.

[15] Xu, Y., Musgrave, Z., Noble, B., and Bailey, M. Bobtail: Avoiding Long Tails In The Cloud. In *NSDI* (2013), 329–342.