On the Feasibility of Measuring the Internet Through Smartphone-based Crowdsourcing

Adriano Faggiani, Enrico Gregori[†], Luciano Lenzini[*], Simone Mainardi[*†], Alessio Vecchio[*]
[†]Dept. of Information Engineering, University of Pisa, Pisa, Italy
firstname.lastname@iet.unipi.it
[*]Institute of Informatics and Telematics (IIT), Italian National Research Council (CNR), Pisa, Italy
firstname.lastname@iit.cnr.it

Abstract—**The large base of Internet-enabled smartphones provides an excellent opportunity for a fine-grained observation of the structure of the Internet and a quantitative evaluation of its characteristics. Smartphones can operate as active mobile monitors and, coordinated by a central entity, they can probe the network on a local scale. Then the results produced by a large number of participants can be merged to obtain a detailed graph of the Internet. Besides the design of the measurement framework, this paper describes the implementation and validation of a traceroute-like tool that is compatible with the Android platform. This confirms that smartphone-based crowdsourcing of network properties can be a viable strategy.**

I. Introduction

An accurate Internet topology graph is important in many areas of networking, from deciding ISP business relationships to diagnosing network anomalies. Most Internet mapping efforts (e.g. [3], [10], [11], [5], [7]) have derived the network structure, at the autonomous system (AS) level, from a limited number of data sources such as Border Gateway Protocol (BGP) paths or traceroute traces. The advantage of using BGP paths is that they can be gathered passively from BGP route collectors and thus require a minimal measurement effort. Connections between BGP neighbors, also called peers, are established by hand, then they communicate to maintain coherent tables of network reach-ability. Unfortunately, the publicly available BGP paths do not cover the entire Internet due to issues such as visibility constraints, route aggregation, hidden sub-optimal paths and policy filtering. In contrast, traceroute measurements provide the ability to infer the data paths that packets take when traversing the Internet. Because they are active measurements, traceroutes can be designed to potentially cover every corner of the Internet given sufficient numbers of vantage points, i.e. locations with distinct network views. To cope with the presence of load-balancing routers, advanced tracing tools, such as Paris traceroute, have been developed in the last few years. Nevertheless, despite significant research work (such as [6], [8], [4], [9], [12], [2]), the structure of the Internet has not yet been fully understood and a large number of links is still to be discovered.

The structure and technical performance of the Internet can be measured in at least two distinct ways: from within the Internet core (top-down, from the ISP side) or from the Internet edges (bottom-up, from the end-user side). Previous research followed the top down approach and was not completely successful because of two major reasons: *i)* measurement campaigns have been carried out with a small number of dedicated observation points (three or four order of magnitude less than the cardinality of the Internet); *ii)* monitors are often statically located in proximity of the Internet core, which often prevents them from observing the edges of the network (where the vast majority of Internet users is located). Furthermore, even if some measurement systems with a large number of monitors have been deployed, they proved to be unable to characterize the Internet accurately because of the not optimal positioning of observation points.

In this paper we propose an innovative measuring technique which operates according to the bottom-up approach. The novel idea is to obtain detailed Internet maps through crowdsourcing, using Internet-enabled smartphones as mobile monitoring nodes.

A. Measuring the Internet through crowdsourcing

To circumvent the drawbacks of the top-down method, we propose a mobile, user-centric, bottom up, energy efficient, and scalable active measurement framework. This can be achieved by running measurement tools in the ever increasing number of mobile smartphones, tablets, and laptops. Probing the Internet both from the bottom up to the inner core towards tier-1 ISPs (vertical probing) and from end-user to end-user (horizontal probing) with a large number of cooperative users distributed all over the world will allow the discovery of the Internet peripheral structure, which is in large part invisible to top-down measurement systems. At the same time, the performance that end-users experience at the fringes of the Internet is put under observation. This kind of information is of paramount importance to end-users, researchers and ISPs.

The system can be used to gather information about the set of routers and ISPs that connect a pair of communicating end-users. At the same time, end-to-end paths can be characterized by the value of several metrics, such as the latency or the minimum available bandwidth. According to this vision, the final graph will be obtained by merging together the outcomes of a large number of micro short-range measurements where the associated mobile monitors inject a negligible quantity of traffic into the Internet. This approach makes the measurement system both scalable with respect to the number of participating smartphones and very efficient from the point

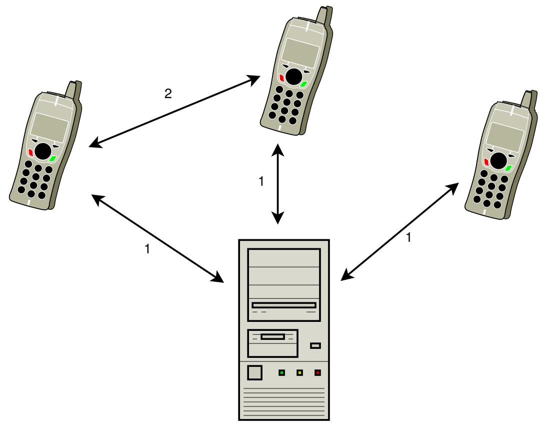

Fig. 1. Overview of the system architecture

of view of energy consumption which is clearly an issue for any mobile terminal. It is important to notice that mobility of monitors provides an additional dimension for the analysis of the Internet, as it enables the evaluation of performance variations (expressed, for instance, in terms of end-to-end delay or minimum bandwidth available along the path) when a mobile user changes his/her position within a cell or when he/she switches the wireless access technology (e.g. from Wi-Fi to HSPA).

II. Overview

The system includes a large set of clients (the smartphones acting as mobile monitors) and a server that orchestrates the measurement campaigns, collects the results, and, after a merging phase, stores them on a database. The server decomposes a measurement campaign into a set of loosely interdependent jobs, that are assigned to clients. Each client executes its measurement job and reports back to the server the obtained results (arrows 1 in Figure 1). Assignment of jobs to clients can be based on their current position. For this reason, each client must periodically update the server about its location. In addition to this client-server protocol, a client-client protocol is used when the measurement job involves multiple endpoints (arrow 2 in Figure 1). In this case the clients coordinate with each other to carry out the end-to-end analysis (e.g. capacity, bottleneck).

The software running on the client-side is devoted to the following major operations: *i)* communication with the central server; *ii)* execution of measurement jobs; *iii)* storage of intermediate results and user preferences; *iv)* presentation to the user of network information that he/she considers useful and valuable, as a form of incentive. The smartphone application also has to meet the following non functional requirements: it must be freely available on the main app markets to maximize the user enrollment and to make possible a large-scale deployment; moreover, the application should silently run on the background and it should take minimum resources from the smartphone. It is also important to ensure that the client functions are perceived as a useful service and they must be presented in a simple but effective way. End users are, in general, not much interested in scientific and technical results, thus to provide a feedback for their efforts a clear user interface and friendly design is mandatory. The client must be available for the major operating systems, e.g., Android and iOS. A corollary effect of distributing the software through app markets is the ease of deployment of updated or upgraded versions.

The server runs on a central machine and represents the mind of the architecture. The main duties of the server are *i)* the management of the various clients, *ii)* the supervision of the measurement campaigns, and *iii)* the storage of the collected measurements. The management of clients consists in an access phase, in which each client has to be identified to join the measurement infrastructure, and in a background location phase, in which the server maintains the information about the geo/net-location of each client and checks for any significant variation. The server needs to identify: *i)* which clients are the best candidates to be involved in a given measurement campaign, *ii)* which are the targets of the campaign (e.g. routers, fixed hosts or other clients) and *iii)* which kind of measurement is required to be run (e.g. capacity estimation, RTT estimation). In some cases, it is possible that the server itself could be involved in some kind of measurements, thus it must also allocate some local resources. Finally, once the measurement campaign is completed, the server has to collect and store the data.

III. System architecture

As mentioned, the measurement system is composed of a server and a large number of mobile devices (clients). The server implements the high-level policies that must be adopted for the measurement campaigns and operates as a centralized repository of collected data. When a new measurement campaign is started, the server decomposes the global task into a possibly large number of jobs that are distributed to the clients. The assignment of jobs to clients can take into account the particular properties of the clients, such as their geographical position or the network they belong to. Each client measures some properties of a small portion of the Internet, as described in the job received from the server, stores locally the intermediate results, and transmits the results back to the server once finished. The server processes and merges the received data to obtain the final graph.

A. Client

Figure 2 depicts a high-level architecture of the client-side software. In the following, the modules that compose the system are described in more detail.

Coordinator. It takes care of coordinating all the client-side activities; this includes communication with the server (receiving new jobs and sending back the results) and scheduling of the measurement duties. Communication and measurement activities should be carried out in background, with little impact on the user's experience.

Analyzer. It provides the measurement tools. It is organized as a collection of subsystems, where each subsystem is able to measure a specific property of the Internet (number of hops

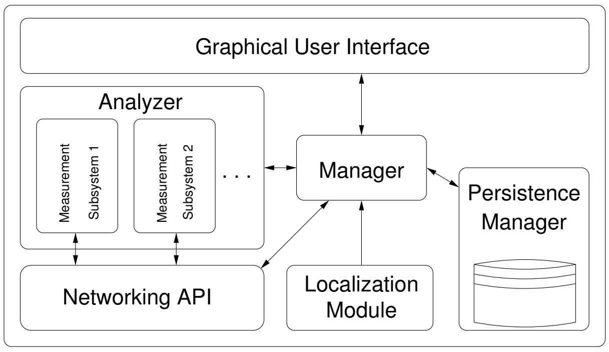

Fig. 2. Client architecture

and IP addresses of the interfaces along the path towards a given target, bandwidth of a link, latency, etc).
Persistence manager. It provides persistent storage for the intermediate results and users preferences.
Graphical user interface. It accomplishes a two-fold purpose: it lets the user define his/her preferences and provides a visual representation of the properties of the network that surrounds him/her (as an incentive to adopt and use the application).
Localization module. The results of the analysis are associated with the position of the client.

B. Server

Figure 3 shows the main components of the software running on the server-side. The modules that compose the server are here described.
Scheduler. It receives the specification of a measurement campaign, divides the global task into a set of jobs, and assigns the jobs to the available clients. This module may use various policies to assign jobs to clients, based on the clients' context and history. The context may include the geographic position (e.g. for analyzing the route towards the farthest point in a country), the client network (e.g. if the client is in a given network then execute a particular job), or its battery level.
Client tracker. It tracks the status and context of clients and logs such information into the database. It also tracks communication history between clients and the server.
Communicator. It handles communication with clients: it sends jobs and receives the results and status updates.
Data manager. It performs consistency checks and merges data into a unique graph.
Data refiner. In case of incomplete information coming from the clients, the server can integrate such data by performing a limited set of measurements. For instance, if a client is not able to execute a specific de-aliasing technique, because of limitations imposed by the OS, this operation can be carried out on the server-side.
Database. It provides persistent storage for all the other modules.

IV. Implementation on the Android platform

We implemented a prototype of the software running on the client-side for the Android platform. The standard Android libraries and programming model provided adequate

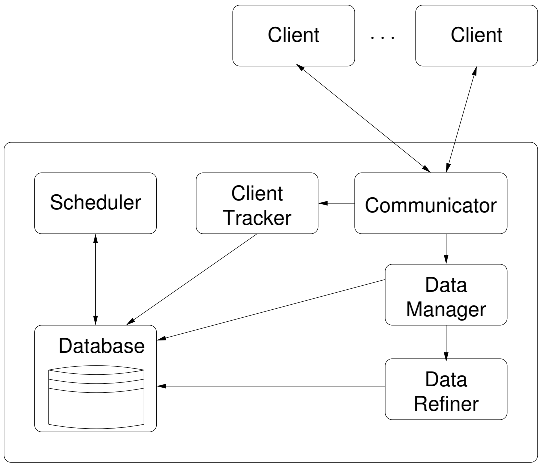

Fig. 3. Server architecture

support for the implementation of the GUI, persistence of data, and communication with the server. Implementation of the measurement subsystems, on the contrary, proved to be a challenging task. The main difficulties have been originated by the networking API available on the Android platform at the socket level, the same of the standard Java language. Such API is characterized by a level of abstraction that is too high with respect to the necessities that arise from the implementation of low-level network measurement mechanisms. In more detail, the absence of raw sockets makes troublesome the manipulation of packet headers and all the data that is not accessible at the application level.

Current implementation is provided with an analyzer module that includes a single measurement subsystem: a traceroute tool that operates similarly to the multipath detection algorithm (MDA) [13], an extension of Paris traceroute [1]. At the moment, our implementation uses only UDP datagrams for estimating the route towards a given host. It is able to recognize load-balancing routers and to enumerate their interfaces.

The basic principles are the same of traceroute, and they are here recalled for the sake of clarity. The source node emits packets, called probes, with increasing TTL values to discover the hosts along the path towards the destination. When the packet arrives on a router its TTL is decremented, and if its value becomes zero the packet is discarded and the router replies with an ICMP Time Exceeded.

This base mechanisms have been extended by tools like Paris traceroute and MDA to cope with load balancing routers, which can forward a packet towards different next-hop interfaces. Per-flow load balancing routers classify incoming packets on the base of some fields of the IP and UDP/TCP packet header (such as source address, destination address, source port, destination port, protocol and checksum). Per-destination load balancing routers forward packets to a given interface only on the base of the destination address. Per-packet load balancers distribute packets evenly over multiple connections and their decisions are not based on the content of packets. Paris traceroute and MDA are able to discover the multiple next-hop interfaces of per-flow load balancers by operating as follows. Once a new router is detected, the source node generates additional probes passing through that router and having different values in the header fields, so that they are classified as belonging to different flows. Then, for every discovered interface, a new set of probes is generated to find its next hop. This last step requires the production of probes having a specific flow id, so that they are all forwarded through the same interface.

In general, a path can be traced by using one of the following protocols: ICMP, UDP, and TCP. An ICMP-based implementation requires access to raw sockets, that are used to receive and analyze the ICMP replies coming from the probed hosts. But to use raw sockets an application must run with the superuser privilege level, and this is not possible on the Android platform where applications are executed as normal users. Thus, we discarded the idea of an ICMP-based implementation. The impossibility of receiving ICMP packets also makes troublesome the implementation of UDP- and TCP-based tracing mechanisms. We found a solution for the UDP-based version by using the IP_RECVERR option of BSD sockets. This option enables extended reliable error message passing: when it is applied to a datagram socket, all generated errors will be queued in a per-socket error queue; then the user can obtain information about the occurred errors by calling the *recvmsg()* function with the MSG_ERRQUEUE flag set. This mechanism can be used to achieve trace-routing functionalities according to the following sequence of operations: the mobile monitor emits a UDP probe datagram with a given value of TTL, say k; the k-th router discards the datagram and replies with a Time Exceeded ICMP error; when the ICMP packet is received on the mobile terminal, the error is stored within the queue associated with the socket, and the next time a primitive is called on that socket an error status is notified to the application level; the application uses the *recvmsg()* call to retrieve information about the error and receives an instance of the *sock_extended_err* structure. This structure contains the type of error occurred, the code and type fields of the ICMP packet, and the address of the sending host. The TTL value of outgoing datagrams can be manipulated by using the *setsockopt()* primitive. Matching between a probe and the corresponding ICMP reply is automatically achieved because all operations take place on the same socket and because a new probe is emitted only after having received the reply for the previous one. The IP_RECVERR option can be set only on datagram sockets, thus this technique cannot be used for a TCP-based implementation of traceroute.

It should be noted that these are standard mechanisms for BSD sockets but they are not available at the Java API level. For this reason, our implementation is split into a Java part and a native part, written in C. The native code directly interacts with the Linux kernel primitives (a Linux 2.6.x kernel is at the base of the Android platform). Mixing Java and native code is made possible by the Android Native Development Kit (NDK).

A. *Parallelization*

The previous solution suffer from the lack of parallelization: an application cannot send a pool of probes and then wait for the replies, it can only send a single probe at a time and wait for the associated ICMP error. To overcome this limitation we enhanced the system with the use of multiple sockets and with different threads of execution. Our implementation makes use of two types of threads, *Breadth Explorer* and *Depth Explorer*. A Breadth Explorer does not directly send probes, it creates a pool of Depth Explorers and waits for the results of their actions. Every Depth Explorer emits a probe and waits for the reply. When the results produced by all the Depth Explorer are ready, the Breadth Explorer aggregates them and update a global data structure where all the found interfaces along a path are stored.

Every Depth Explorer generates probes having a different source port, as it uses a different socket. It also sends the probes to a destination address that is adjacent to the address of the target, and that is different from the one used by its siblings. The use of adjacent addresses makes possible the discovery of next-hop interfaces in the presence of routers that adopt per-destination load balancing policies. Moreover, since the source port is also changed, the probes are considered as having a different flow id if a per-flow load balancer is found.

On receiving a reply, every Depth Explorer checks if the interface is a new one or if it is already known to the system. If the interface is a new one, it is added as a next-hop of the interface currently under examination; then, if the maximum number of probes has not already been reached[1], six new Depth Explorers are started.

B. *Classification of the load-balancing policy*

After the discovery phase, the system detects the policy adopted by load-balancing routers (if it is per-flow, per-destination, or per-packet). This is done by sending probes having a fixed destination address and changing the flow id (by using different source port numbers). If the replies are all generated by the same interface, then the load-balancing policy is classified as per-destination. On the contrary, if the replies arrive from different interfaces, then the mobile monitor sends an additional set of probes having the same flow id (using sequentially a single socket): if the replies associated to this probes arrive from the same interface then the load-balancer is classified as per-flow, otherwise as per-packet.

C. *Limitations*

One of the limitations of this technique is the absence of information useful to build de-aliasing mechanisms (on Android mobile terminals): since the content of ICMP packets cannot be fully analyzed by the application level, all fingerprint-based de-aliasing algorithm (e.g. MIDAR) cannot be used. However, path information provided by Android terminals can be post-processed on the server where a de-aliasing module can be implemented using state of the art techniques (as mentioned in the previous section, the Data refiner module aims at overcoming the limitations that are peculiar of specific device families).

[1]The maximum number is equal to 96 as suggested by [13]

V. Experiments

We experimentally validated and evaluated the implementation of the traceroute mechanisms for the Android platform through a series of measurements. The Android device from which the traceroutes were sent was connected to the Internet via a Wi-Fi access point.

A. Selection of Target Destinations

We chose 141 target destinations belonging to the GARR[2], the Italian network that connects universities and research centers. We decided to select the targets within this network because its topology is freely available to researchers (through the GARR website). Information about topology and status can be browsed through the GARR Integrated Networking Suite[3] (GINS); information available includes router interfaces, traffic, load percentages – regarding the backbone of the network, as well as its users and Point-of-Presence (PoP) in the territory. We used this system to retrieve the list of IP addresses associated to target destinations: all the Italian *i)* libraries; *ii)* public (private) universities; and *iii)* Italian National Research Council (CNR) centers connected to GARR PoPs.

We executed traceroutes to these destinations from a facility of the CNR located in Pisa. As it can be verified by checking the GINS[4], this network is directly connected to the GARR PoP of Pisa (PoP-PI1, `rt.pi1.garr.net`). Accordingly, since our gateway router is directly connected to this PoP, only the first interface discovered in our probes belongs to the CNR network of Pisa. The others, reachable in a number of hops greater than or equal to 2, are always within the GARR network. Specifically, we detected 21 GARR network interfaces, all with addresses included in the subnet 193.206.128.0/20. Who-is queries confirm that this subnet is assigned to the GARR for its backbone and PoPs.

B. Validation

We carried out the validation by: *i)* comparing the interfaces discovered with those made available by the consortium managing the GARR; and by *ii)* comparing our results with those produced by a popular traceroute tool.

For each GARR network interface detected, we browsed the GINS to check whether that interface belongs to a router. We found that the totality of the interfaces are within PoP or are associated with backbone routers. Moreover, for each pair of interfaces belonging to the same path, we found that they belong to physically adjacent routers. Therefore, we can say that our tool succeeds in discovering GARR network interfaces.

The results produced by our tools have been compared with the ones produced by Paris traceroute, which is open-source, freely-available[5], and it is considered as one of the most reliable tools for tracing paths. We executed Paris traceroute on a Linux PC connected to the same Wi-Fi access point of the Android device. In order to make it behave exactly like our tool – i.e. classifying load balancing routers and finding all the interfaces for each hop – we decided to run Paris traceroute with the `--algo=exhaustive` option. However, by looking at the source code, we observed that the implementation of the functionalities needed for the detection of routers adopting per-destination load balancing policies was still incomplete. Thus, to obtain compliant and comparable results, we turned this feature off also in our tool. Finally, we set a timeout of 1000 ms for each probe. The results confirmed the correctness of our solution: Paris traceroute and the Android-based implementation discovered exactly the same set of interfaces and the same path for all the destinations.

C. Evaluation

Experimental evaluation has been accomplished as follows. First, we determined to what extent the interfaces detected in our measurements are also reported by a widely spread, distributed traceroute framework. Then, we analyzed the characteristics of the paths followed by our probes and captured the number of packets sent, in order to determine the total amount of IP traffic generated. Finally, we monitored battery consumption of the device during the experiments.

The Cooperative Association for Internet Data Analysis (CAIDA) collects several types of Internet properties at geographically and topologically different locations, and makes this data available to the research community. We decided to use October 2011 CAIDA Macroscopic Internet Topology Data Kit (ITDK) to evaluate our tool. ITDK contains two Internet router-level topologies, produced from active measurements conducted on the Archipelago (Ark) measurement infrastructure (the focus of Ark is coordinating large-scale traceroute-based topology measurements). The two topologies considered, which differ only on the algorithms used for associating interfaces to physical routers, contain almost all the the interfaces we detected. We registered two exceptions: the first interface which is not present in the ITDK topologies has IP address 146.48.99.254, which corresponds to our gateway in the CNR network; the second has IP address 193.206.136.29, which corresponds to the GARR PoP-PI1 router interface directly connected to the CNR of Pisa. In order to understand the reasons behind these CAIDA missed detections, we searched for missed interfaces within couples of consecutive IP addresses within ITDK topologies. The rationale behind this choice is that if you traverse a point-to-point link, you will almost surely detect only one interface – the one which gets you out from that link. Therefore, the detected interface depends on the direction a given link is passed through. For the point-to-point link between the CNR of Pisa and the GARR PoP-PI1, we find the interface with IP address 193.206.136.30 in the ITDK topologies. This is consistent since our tool traversed that link from the CNR to the GARR PoP-PI1, while CAIDA traceroutes traversed it in the opposite direction. From these observations it follows that a large-scale deployment of our tool between end-users connected to non-backbone networks, could be fundamental to traverse links in both directions and discover all their interfaces.

[2]http://www.garr.it

[3]http://ginsdr.dir.garr.it

[4]http://ginsdr.dir.garr.it/Weathermap/mapgen_auto.php?type=eql3&id=rt.pi1.garr.net

[5]http://www.paris-traceroute.net/

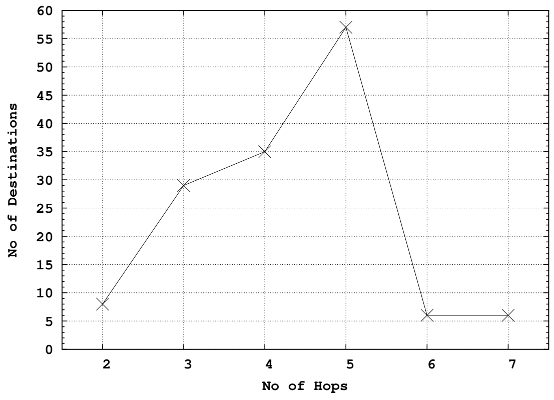

Fig. 4. Number of destinations vs. number of hops required to reach them

We now focus on the volume of traffic generated by our application. Since this volume depends on the number of hops needed to reach a given destination, we first calculated (Fig. 4) the number of probed destinations reachable with a given number of hops. The average number of hops is 4.30. Therefore, chosen destinations are quite close to our device. In practice, if destinations are chosen at random, the number of hops required to reach them could be higher. Our application sent a total of 110,539 UDP datagrams with no payload to probe the whole set of destinations. Since UDP header is 8 Bytes in size and each datagram is encapsulated in an IP packet, the total volume of IP traffic generated is approximately 3.16 MBytes (IP header is 24 Bytes).

We conclude the evaluation with a preliminary study about energy consumption. We measured the battery level of our device during the experiments by implementing and registering an Android `BroadcastReceiver`. This allows the application-level to receive `ACTION_BATTERY_CHANGED` broadcasts from the Android system and to retrieve the `EXTRA_LEVEL` information from the `BatteryManager` API. We repeated the measurements several times and we never measured battery level reductions greater than 1%. Therefore, we can reasonably say that our application can be executed by end-users with extremely low impact on their device battery and that energy consumption, a major problem in the context of mobile device, is not an impediment for smartphone-based crowdsourcing of Internet measures.

VI. Conclusion and future work

A thorough understanding of the Internet requires detailed information about its topology and its performance figures. Given the size of the Internet, an approach based on crowdsourcing may fit the scope in an unprecedented way: the results of a large number of short-range measurements, carried out using currently available smartphones, can be combined together to generate a fine-grained map of the network. Besides their potentially huge number, the use of smartphones as network monitors provides other opportunities: *i)* performance is observed at the periphery of the network, where the majority of end-users is located, *ii)* mobility of terminals allows the monitoring system to collect dynamical and geo-referenced information.

The major contributions of this paper are the following. The idea of gathering information about the Internet through smartphone-based crowdsourcing is novel and it is here proposed for the first time (as far as we know). The paper describes an architecture for a measurement system that includes from the beginning the use of mobile monitors. The implementation of a prototype, provided with a measurement subsystem that operates similarly to Paris traceroute, demonstrates that the proposed idea can be put into practice, despite the limitations of mobile operating systems. The preliminary evaluation of the energy consumption on the client side shows that the application can be compatible with a satisfying user experience.

Future work will focus on the implementation of additional measurement modules and on porting the client to other popular operating systems (such as iOS). At the same time, we are developing a module useful to perform address de-aliasing on the server. This will allow to refine the path information generated by Android devices (preliminary analysis seems to show that this problem does not arise on iOS-based devices, which will be able to carry out de-aliasing techniques directly on the client-side).

References

[1] Brice Augustin, Timur Friedman, and Renata Teixeira. Multipath tracing with Paris traceroute. In *Proceedings of the Fifth IEEE/IFIP Workshop on End-to-End Monitoring Techniques and Services (E2EMON 07)*, pages 1–8. IEEE, 2007.

[2] Zachary S. Bischof, John S. Otto, Mario A. Sánchez, John P. Rula, David R. Choffnes, and Fabián E. Bustamante. Crowdsourcing ISP characterization to the network edge. In *Proceedings of the first ACM SIGCOMM workshop on measurements up the stack (W-MUST 11)*, pages 61–66. ACM, 2011.

[3] H. Chang, R. Govindan, S. Jamin, S. J. Shenker, and W. Willinger. Towards capturing representative AS-level Internet topologies. *Computer Networks*, 44(6):737–755, 2004.

[4] Kai Chen, David R. Choffnes, Rahul Potharaju, Yan Chen, Fabian E. Bustamante, Dan Pei, and Yao Zhao. Where the sidewalk ends: extending the Internet as graph using traceroutes from P2P users. In *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies (CoNEXT 09)*, pages 217–228. ACM, 2009.

[5] R. Cohen and D. Raz. The Internet Dark Matter: on the Missing Links in the AS Connectivity Map. In *IEEE INFOCOM*, 2006.

[6] Cooperative Association for Internet Data Analysis (CAIDA). Internet topology data kit.

[7] Yihua He, Georgos Siganos, Michalis Faloutsos, and Srikanth Krishnamurthy. A systematic framework for unearthing the missing links: measurements and impact. In *Proceedings of the 4th USENIX conference on networked systems design and implementation (NSDI 07)*, pages 14–14. USENIX Association, 2007.

[8] B. Huffaker, A. Dhamdhere, M. Fomenkov, and k. Claffy. Toward Topology Dualism: Improving the Accuracy of AS Annotations for Routers. In *Passive and Active Network Measurement Workshop (PAM 10)*, Zurich, Switzerland, April 2010.

[9] Ken Keys. Internet-scale IP alias resolution techniques. *SIGCOMM Comput. Commun. Rev.*, 40:50–55, January 2010.

[10] Ricardo V. Oliveira, Dan Pei, Walter Willinger, Beichuan Zhang, and Lixia Zhang. In search of the elusive ground truth: the internet's AS-level connectivity structure. *SIGMETRICS Perform. Eval. Rev.*, 36:217–228, June 2008.

[11] Ricardo V. Oliveira, Beichuan Zhang, and Lixia Zhang. Observing the evolution of Internet AS topology. *SIGCOMM Comput. Commun. Rev.*, 37:313–324, August 2007.

[12] Yuval Shavitt and Eran Shir. DIMES: let the Internet measure itself. *SIGCOMM Comput. Commun. Rev.*, 35:71–74, October 2005.

[13] Darryl Veitch, Brice Augustin, Renata Teixeira, and Timur Friedman. Failure control in multipath route tracing. In *INFOCOM*, pages 1395–1403. IEEE, 2009.